ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MANUSCRIPT-BASED THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph.D.

BY
Wael KHREICH

TOWARDS ADAPTIVE ANOMALY DETECTION SYSTEMS USING BOOLEAN COMBINATION OF HIDDEN MARKOV MODELS

MONTREAL, JULY 18, 2011

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS :

Mr. Éric Granger, Thesis Supervisor
Département de génie de la production automatisée at École de technologie supérieure

Mr. Robert Sabourin, Thesis Co-supervisor
Département de génie de la production automatisée at École de technologie supérieure

Mr. Ali Miri, Thesis Co-supervisor
School of Computer Science, Ryerson University, Toronto, Canada

Mr. Jean-Marc Robert, President of the Board of Examiners
Département de génie logiciel at École de technologie supérieure

Mr. Tony Wong, Examiner
Département de génie de la production automatisée at École de technologie supérieure

Mr. Qinghan Xiao, External examiner
Defence Research and Development Canada (DRDC), Ottawa

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

ON JUNE, 21 2011

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I would like to express sincere appreciation to Professor Éric Granger my research supervisor for his worthwhile assistance, support, and guidance throughout this research work. In particular, I would like to thank him for his protracted patience during those many hours of stimulating discussion. In addition, the progress I have achieved in writing is mostly due to his thorough and critical reviews of my manuscripts.

I would like to extend my heartfelt gratitude to my co-supervisor Professor Ali Miri whose constant encouragement and valuable insights provided the perfect guidance that I needed through those years of research. I am deeply indebted to Prof. Ali Miri, who first believed in me and introduced me to Prof. Éric Granger and made possible the first step of this PhD thesis.

I would like to express my sincere and special thanks to Professor Robert Sabourin my co-director, for his support and instruction throughout this project. Each meeting with him added invaluable aspects to the project and broadened my perspective. His stimulating discussions and comments led to much of this work.

My gratitude also goes to members of my thesis committee: Prof. Jean-Marc Robert, Prof. Tony Wong, and Prof. Qinghan Xiao for their time and their constructive comments.

Thanks to all the wonderful people that I met at LIVIA and that made all this experience so satisfactory. Special thanks to Dominique Rivard, Eduardo Vellasques, Éric Thibodeau, Luana Batista, Marcelo Kapp, Paulo Cavalin, Vincent Doré, and Youssouf Chherawala.

Thanks to all my friends back home, who made me feel like I never left and provided me with long-distance support. Thanks to my friends Joy Khoriaty, Toufic Khreich and Myra Sader, for their moral support and help. Special thanks to Marc Hasrouny, Patrick

Diab, Sam Khreiche, Roger Mehanna, and Julie Barakat who have supported me since I came to Canada, and made me feel like home.

To my beloved wife Roula, thank you for being by my side, for putting up with all those nights and weekends that I spent in front of my laptop, and for giving me endless support and motivation to finish this thesis. Your being here was the best thing that could have happened to me. Thank you.

Towards the end, gratitude goes to my loved mother and father back home for their endless love and sacrifices, and for their sustained moral and financial support during those years of research and throughout my whole life. You were always my source of strength.

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and le Fonds québécois de la recherche sur la nature et les technologies (FQRNT). The author gratefully acknowledges their financial support.

TOWARDS ADAPTIVE ANOMALY DETECTION SYSTEMS USING BOOLEAN COMBINATION OF HIDDEN MARKOV MODELS

Wael KHREICH

ABSTRACT

Anomaly detection monitors for significant deviations from normal system behavior. Hidden Markov Models (HMMs) have been successfully applied in many intrusion detection applications, including anomaly detection from sequences of operating system calls. In practice, anomaly detection systems (ADSs) based on HMMs typically generate false alarms because they are designed using limited representative training data and prior knowledge. However, since new data may become available over time, an important feature of an ADS is the ability to accommodate newly-acquired data incrementally, after it has originally been trained and deployed for operations. Incremental re-estimation of HMM parameters raises several challenges. HMM parameters should be updated from new data without requiring access to the previously-learned training data, and without corrupting previously-learned models of normal behavior. Standard techniques for training HMM parameters involve iterative batch learning, and hence must observe the entire training data prior to updating HMM parameters. Given new training data, these techniques must restart the training procedure using all (new and previously-accumulated) data. Moreover, a single HMM system for incremental learning may not adequately approximate the underlying data distribution of the normal process, due to the many local maxima in the solution space. Ensemble methods have been shown to alleviate knowledge corruption, by combining the outputs of classifiers trained independently on successive blocks of data.

This thesis makes contributions at the HMM and decision levels towards improved accuracy, efficiency and adaptability of HMM-based ADSs. It first presents a survey of techniques found in literature that may be suitable for incremental learning of HMM parameters, and assesses the challenges faced when these techniques are applied to incremental learning scenarios in which the new training data is limited and abundant. Consequently, An efficient alternative to the Forward-Backward algorithm is first proposed to reduce the memory complexity without increasing the computational overhead of HMM parameters estimation from fixed-size abundant data. Improved techniques for incremental learning of HMM parameters are then proposed to accommodate new data over time, while maintaining a high level of performance. However, knowledge corruption caused by a single HMM with a fixed number of states remains an issue. To overcome such limitations, this thesis presents an efficient system to accommodate new data using a learn-and-combine approach at the decision level. When a new block of training data becomes available, a new pool of base HMMs is generated from the data using a different number of HMM states and random initializations. The responses from the newly-trained HMMs are then combined to those of the previously-trained HMMs in receiver operat-

ing characteristic (ROC) space using novel Boolean combination (BC) techniques. The learn-and-combine approach allows to select a diversified ensemble of HMMs (EoHMMs) from the pool, and adapts the Boolean fusion functions and thresholds for improved performance, while it prunes redundant base HMMs. The proposed system is capable of changing its desired operating point during operations, and this point can be adjusted to changes in prior probabilities and costs of errors.

During simulations conducted for incremental learning from successive data blocks using both synthetic and real-world system call data sets, the proposed learn-and-combine approach has been shown to achieve the highest level of accuracy than all related techniques. In particular, it can sustain a significantly higher level of accuracy than when the parameters of a single best HMM are re-estimated for each new block of data, using the reference batch learning and the proposed incremental learning techniques. It also outperforms static fusion techniques such as majority voting for combining the responses of new and previously-generated pools of HMMs. Ensemble selection techniques have been shown to form compact EoHMMs for operations, by selecting diverse and accurate base HMMs from the pool while maintaining or improving the overall system accuracy. Pruning has been shown to prevents pool sizes from increasing indefinitely with the number of data blocks acquired over time. Therefore, the storage space for accommodating HMMs parameters and the computational costs of the selection techniques are reduced, without negatively affecting the overall system performance. The proposed techniques are general in that they can be employed to adapt HMM-based systems to new data, within a wide range of application domains. More importantly, the proposed Boolean combination techniques can be employed to combine diverse responses from any set of crisp or soft one- or two-class classifiers trained on different data or features or trained according to different parameters, or from different detectors trained on the same data. In particular, they can be effectively applied when training data is limited and test data is imbalanced.

Keywords: Intrusion Detection Systems, Anomaly Detection, Adaptive Systems, Ensemble of Classifiers, Information Fusion, Boolean Combination, Incremental Learning, On-Line Learning, Hidden Markov Models, Receiver Operating Characteristics.

VERS DES SYSTÈMES ADAPTATIFS DE DÉTECTION D'ANOMALIES UTILISANT DES COMBINAISONS BOOLÉENNES DE MODÈLES DE MARKOV CACHÉS

Wael KHREICH

RÉSUMÉ

La détection d'anomalies permet de surveiller les déviations significatives du comportement normal d'un système. Les modèles de Markov cachés (MMCs) ont été utilisés avec succès dans différentes applications de détection d'intrusions, en particulier la détection d'anomalies à partir de séquences d'appels système. Dans la pratique, les systèmes de détection d'anomalies (SDAs) basés sur les MMCs génèrent typiquement de fausses alertes étant donné qu'ils ont été conçu en utilisant des données d'apprentissage et des connaissances préalables limitées. Mais puisque de nouvelles données peuvent être acquises avec le temps, les SDAs doivent s'adapter aux nouvelles données une fois qu'ils ont été entraînés et mis en opération. La ré-estimation incrémentale des paramètres des MMCs présente plusieurs défis à relever. Ces paramètres devront être ajustés selon l'information fournie par les nouvelles données, sans réutiliser les données d'apprentissage antérieures et sans compromettre les informations déjà acquises dans les modèles de comportement normal. Les techniques standards pour l'entraînement des paramètres des MMCs utilisent l'apprentissage itératif en mode « batch » et doivent ainsi observer la totalité de la base de données d'apprentissage avant d'ajuster les paramètres des MMCs. À l'acquisition de nouvelles données, ces techniques devraient recommencer la procédure de l'apprentissage en utilisant toutes les données accumulées. En outre, un système d'apprentissage incrémental basé sur un seul MMC n'a pas la capacité de fournir une bonne approximation de la distribution sous-jacente du comportement normal du processus à cause du grand nombre des maximums locaux. Les ensembles de classificateurs permettent de réduire la corruption des connaissances acquises, en combinant les sorties des classificateurs indépendamment entraînés sur des blocs de données successives.

Cette thèse apporte des contributions au niveau des MMCs ainsi qu'en regard de la décision des classificateurs dans le but d'améliorer la précision, l'efficacité et l'adaptabilité des SDAs basés sur les MMCs. Elle présente en premier lieu une étude d'ensemble des techniques qui peuvent être utilisées pour l'apprentissage incrémental des paramètres des MMCs et évalue les défis rencontrés par ces techniques lors d'un apprentissage incrémental avec des données limitées ou abondantes. Par conséquent, une alternative efficace à « l'algorithme avant-arrière » est proposée pour réduire la complexité de la mémoire sans augmenter le coût computationnel de l'estimation des paramètres des MMCs, faite à partir de données fixes. D'autre part, elle propose des améliorations aux techniques d'apprentissage incrémental des paramètres des MMCs pour qu'elles s'adaptent à des nouvelles données, tout en conservant un niveau de performance élevé. Cependant, le problème de la corruption des connaissances acquises causée par l'utilisation d'un sys-

tème basé sur un seul MMC n'est toujours pas complètement résolu. Pour surmonter ces problèmes, cette thèse présente un système efficace qui s'adapte aux nouvelles données, en utilisant une approche apprentissage-combinaison au niveau décision. À l'arrivée d'un nouveau bloc de données d'apprentissage, un ensemble de MMCs est généré en variant le nombre d'états et l'initialisation aléatoire des paramètres. Les réponses des MMCs entraînés sur les nouvelles données sont combinées avec celles des MMCs déjà entraînés sur les données antérieures dans l'espace ROC (caractéristique de fonctionnement du récepteur, receiver operating characteristic), en utilisant les techniques de combinaison booléenne. L'approche apprentissage-combinaison proposée permet de sélectionner un ensemble diversifié de MMCs et d'ajuster les fonctions booléennes et les seuils de décision, tout en éliminant les MMCs redondants. Pendant la phase d'opération, le système proposé est capable de changer le point d'opération et celui-ci peut s'adapter au changement des probabilités a priori et des coûts des erreurs.

Les résultats des simulations obtenus sur des bases de données réelles et synthétiques montrent que l'approche apprentissage-combinaison proposée a atteint le taux le plus élevé de précision en comparaison avec d'autres techniques d'apprentissage incrémental à partir de blocs de données successives. En particulier, le système a démontré qu'il pouvait maintenir un plus haut taux de précision que celui d'un système basé sur un seul MMC entraîné selon le mode batch (utilisant tous les blocs de données cumulatifs) ou qui adapte les paramètres du MMC selon la méthode incrémentale proposée (utilisant des blocs de données successifs). De même, la précision du système proposé a surpassé celle des techniques basées sur les fonctions de fusion classiques tel que le vote majoritaire combinant les décisions des MMCs entraînés sur les anciens et les nouveaux blocs de données. Les techniques de sélection d'ensembles du système proposé sont capables de choisir un ensemble compact, en sélectionnant les MMCs générés, les plus précis et les plus diversifiés, tout en conservant ou en améliorant la précision du système. La technique d'élimination des modèles empêche l'augmentation de la taille du pool avec le temps. Ceci permet de réduire l'espace de stockage des MMCs et le coût computationnel des techniques de sélection, sans compromettre la performance globale du système. Les techniques proposées sont générales et peuvent être utilisées dans diverses applications pratiques qui nécessitent l'adaptation aux nouvelles données des systèmes basés sur les MMCs. Qui plus est, les techniques de combinaison booléennes proposées sont capables de combiner les réponses des classificateurs binaires ou probabilistes pour des problèmes à une ou deux classes. Ceci inclut la combinaison du même type de classificateurs entraînés sur différentes données ou caractéristiques ou selon différents paramètres, et la combinaison de différents classificateurs entraînés sur la même de données. En particulier, ces techniques peuvent être efficaces dans des applications où les données d'apprentissage sont limitées et les données de test sont non équilibrées.

Mots-clés : Systèmes de détection d'intrusions, détection d'anomalies, systèmes adaptatifs, ensembles de classificateurs, fusion de l'information, combinaison booléenne, apprentissage incrémental, apprentissage en-ligne, modèles de Markov cachés, caractéristique de fonctionnement du récepteur.

CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

XXII

LIST OF ABBREVIATIONS

μ-HMMs	Multiple Hidden Markov Models.
ADS	Anomaly Detection System.
AS	Anomaly Size.
AUC	Area Under the ROC Curve.
AUCH	Area under the ROC ROC Convex Hull.
BC	Boolean Combination.
BCM	Boolean Combination of Multiple Detectors.
BKS	Behavior Knowledge Space.
BW	Baum-Welch.
CIA	Confidentiality, Integrity and Availability.
CRE	Conditional Relative Entropy.
DMZ	Demilitarized Zone.
DW	Detector Window Size.
EFFBS	Efficient Forward Filtering Backward Smoothing.
ELS	Extended Least Squares.
EM	Expectation-Maximization.
EoC	Ensemble of Classifiers.
EoHMMs	Ensemble of Hidden Markov Models.
FB	Forward Backward.

FFBS	Forward Filtering Backward Smoothing.
FO	Forward Only.
FSA	Finite State Automaton.
GAM	Generalized Alternating Minimization.
GD	Gradient Descent.
GEM	Generalized Expectation-Maximization.
HIDS	Host-based Intrusion Detection System.
HMM	Hidden Markov Model.
IBC	Iterative Boolean Combination.
IBW	Incremental Baum-Welch.
IDPS	Intrusion Detection and Prevention System.
IDS	Intrusion Detection System.
IPS	Intrusion Prevention System.
KL	Kullback-Leibler.
LAN	local Area Network.
LT	Life Time Expectancy of Models.
LCR	Largest Concavity Repair.
MDI	Minimum Discrimination Information.
MLE	Maximum Likelihood Estimation.
MMD	Minimum Model Divergence.

MMI	Maximum Mutual Information.
MM	Markov Model.
MMSE	Minimum Mean Square Error.
MPE	Minimum Prediction Error.
MRROC	Maximum Realizable ROC.
NIDS	Network-based Intrusion Detection System.
NIST	National Institute of Standards and Technology.
ODE	Ordinary Differential Equation.
OS	Operating System.
RCLSE	Recursive Conditioned Least Squares Estimator.
RIPPER	Repeated Incremental Pruning to Produce Error Reduction.
ROCCH	ROC Convex Hull.
ROC	Receiver Operating Characteristic.
RPE	Recursive Prediction Error.
RSPE	Recursive State Prediction Error.
SSH	Secure Shell.
SSL	Secure Sockets Layer.
STIDE	Sequence Time-Delay Embedding.
TIDE	Time-Delay Embedding.
UNM	University of New Mexico.

VPN	Virtual Private Network.
WMW	Wilcoxon-Mann-Whitney.
acc	Accuracy.
bf	Boolean function.
err	Error Rate.
fnr	False Negative Rate.
fpr	False Positive Rate.
setuid	Set user identifier permission.
tnr	True Negative Rate.
tpr	True Positive Rate.
uid	User identifiers.

LIST OF SYMBOLS

Δ	Fixed positive lag in fixed-lag smoothing estimation.
Λ	HMM parameter space.
Σ	Alphabet size of Markov model generator (or of a process).
$\nabla_{(Y)}X$	Gradient of X with reference to Y.
$\alpha_t(i)$	The forward variable. Unnormalized joint probability density of the state i and the observations up to time t, given the HMM λ.
$\beta_t(i)$	The backward variable. Conditional probability density of the observations from time $t+1$ up to the last observation, given the state at time t is i.
$\gamma_{\tau\|t}(i)$	Conditional state density. Probability of being in state i at time τ given the HMM λ and the observation sequence $o_{1:t}$. It is a filtered state densisty if $\tau = t$; predictive state density if $\tau < t$, and smoothed state density if $\tau > t$.
$\delta_i(j)$	Kronecker delta. It is equal to one if $i = j$ and zero otherwise.
ϵ	Small positive constant representing tolerance in error or accuracy measure values.
ε	Output prediction error, $\varepsilon = o_t - \hat{o}_t$.
$\mathcal{E}_t$	HMM output prediction error, $\mathcal{E}_t = o_t - \hat{o}_{t\|t-1}$
ζ	Gradient of state variable *alpha* w.r.t. HMM state transitions.
η	Fixed learning rate.
η_t	Time-varying learning rate.
λ	HMM parameters, $\lambda = (A, B, \pi)$.
$\hat{\lambda}$	Estimated HMM parameters.
λ^0	Initial estimation (or guess) of HMM parameters.
$\hat{\lambda}^k$	Estimated HMM parameters at the $k-$th iteration.
λ_r	Estimated HMM parameters from the r-th observation.
λ_t	Estimated HMM parameters at time t.
μ	Mean of output probability distribution of a continuous HMM.

$\xi_{\tau\|t}(i,j)$	Conditional joint state density. Probability of being in state i at time τ and transiting to state j at time $\tau+1$ given the HMM λ and the observation sequence $o_{1:t}$. It is a filtered joint state densisty if $\tau = t$; predictive joint state density if $\tau < t$, and smoothed joint state density if $\tau > t$.
π	Vector of initial state probability distribution.
$\pi(i)$	Element of π. Probability of being in state i at time $t = 0$.
σ^t	Variance of output probability distribution of a continuous HMM.
$\sigma_t(i,j,k)$	Probability of having made a transition from state i to state j, at some point in the past, and of ending up in state k at the current time t.
ς	Gradient of state variable *alpha* w.r.t. HMM outputs.
$\psi_t(i)$	Gradient of the prediction error cost function (J) w.r.t. HMM parameters.
ω	Decay function.
A	Matrix of HMM state transition probability distribution.
$AUCH_{0.1}$	Partial area under the ROC convex hull for the range of $fpr = [0, 0.1]$.
B	Matrix of HMM state output probability distribution.
D	Block of training data.
D_k	Block of training data received at time $t = k$.
$E[.]$	Expectation function.
E	Selected ensemble of base HMMs from the pool.
E_k	Selected ensemble of base HMMs after receiving the k^{th} block of data.
I	Number of iterations of the IBC algorithm.
$J(\lambda)$	Prediction error cost function based on HMM (λ)
M	Size of output alphabet (V) of a discrete HMM. The number of distinct observable symbols.
N	Number of HMM states.
$N_{ijk}^{t,\tau}(O)$	The probability of making at time t a transition from state i to j with symbol O_t and to be in state k at time $\tau \geq t$.
O	Sequence of observation symbols.
$\mathcal{P}$	Pool of base HMMs.
$\mathcal{P}_k$	Pool of base HMMs generated after receiving the k^{th} block of data.

S	State space of HMM.
$\mathcal{S}$	Selected set of decision thresholds (from each base HMM) and Boolean functions that most improves the overall ROC convex hull of Boolean combination.
T	Length of the observation sequence.
T	Decision threshold.
$\mathcal{T}$	Test data set.
V	Output alphabet of HMM. It can be either a continuous or discrete.
$\mathcal{V}$	Validation data set.
a_{ij}	Element of A. Probability of being in state i at time t and going to state j at time $t+1$.
$a_{ij}^{\tau}(o_t)$	Probability to make a transition from state i to state j and to generate the output o_t at time τ.
$b_j(k)$	Element of B. Probability of emitting an observation symbol o_k at state j.
$\ell_T(\lambda)$	Log-likelihood of the observation sequence $o_{1:T}$ with regard to HMM parameters (λ).
o_t	Observation symbol at time t.
$o_{1:t}$	Concise notation denoting the observation sub-sequence $o_1, o_2, \ldots, o_t$.
$\hat{o}_t$	Output prediction at time t.
$\hat{o}_{t\|t-1}$	Output prediction at time t, conditioned on all previous output values.
q_t	State of HMM at time t. $q_t = i$ denotes that the state of the HMM at time t is S_i.
w_t	Gradient of the state prediction filter ($\gamma_{t\|t-1}$) w.r.t. HMM parameters.

XXX

INTRODUCTION

Computers and network touch every facet of modern life. Our society is increasingly dependent on interconnected information systems, offering more opportunities and challenges for computer criminals to break into these systems. Security attacks through Internet have proliferated in recent years. Information security has therefore become a critical issue for all – governments, organizations and individuals.

As systems become ever more complex, there are always exploitable weaknesses that are inherent to all stages of system development life cycle (from design to deployment and maintenance). Several security attacks have emerged because of vulnerabilities in protocol design. Programming errors are unavoidable, even for experienced programmers. Developers often assume that their products will be used under expected conditions. Many security vulnerabilities are also caused by system misconfiguration and mismanagement. In fact, managing system security is a time-consuming and challenging task. Different complex software components, including operating systems, firmware, and applications, must be configured securely, updated, and continuously monitored for security. This requires a large amount of time, energy, and resources to keep up with the rapidly evolving pace of threat sophistication.

Preventive security mechanisms such as firewalls, cryptography, access control and authentication are deployed to stop unwanted or unauthorized activities before they actually cause damage. These techniques provide a security perimeter but are insufficient to ensure reliable security. Firewalls are prone to misconfiguration, and can be circumvented by redirecting traffic or using encrypted tunnels (Ingham and Forrest, 2005). With the exception of one-time pad cipher, all cryptographic algorithms are theoretically vulnerable to cryptanalytic attacks. Furthermore, viruses, worms, and other malicious code may defeat crypto-systems by secretly recording and transmitting secret keys residing in a computer memory. Regardless of the preventive security measures, incidents are likely to happen. Insider threats are invisible to perimeter security mechanisms. At-

tacks perpetrated by insiders are very often more damaging because they understand the organization business and have access privileges, which facilitate breaking into systems and extracting or damaging critical information (Salem et al., 2008). Furthermore, social engineering bypasses all prevention techniques and provides a direct access to systems. In addition, organizational security policies typically attempt to maintain an appropriate balance between security and usability, which makes it impossible for an operational system to be completely secure.

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity and availability (CIA), or to bypass the security mechanisms of a computer or network (Scarfone and Mell, 2007). It is more feasible to prevent some attacks and detect the rest than to try to prevent everything. When perimeter security fails, intrusion attempts must be detected as soon as possible to limit the damage and take corrective measures, hence the need for an intrusion detection system (IDS) as a second line of defense (McHugh et al., 2000). IDSs monitor computer or network systems to detect unusual activities and notify the system administrator. In addition, IDSs should also provide relevant information for post-attack forensics analysis, and should ideally provide reactive countermeasures. Without an IDS, security administrators may have no sign of many ongoing or previously-deployed attacks. For instance, attacks that are not intended to damage or control a system, but instead to compromise the confidentiality or integrity of the data (e.g., by extracting or altering sensitive information), would be very difficult to detect. An IDS is not a stand-alone system, but rather a fundamental technology that complements preventive techniques and other security mechanisms.

Since their inceptions in the 1980s (Anderson, 1980; Denning, 1986), IDSs have received increasing research attention (Alessandri et al., 2001; Axelsson, 2000; Debar et al., 2000; Estevez-Tapiador et al., 2004; Lazarevic et al., 2005; Scarfone and Mell, 2007; Tucker et al., 2007). IDSs are classified based on their monitoring scope into host-based IDS (HIDS) and network-based IDS (NIDS). HIDSs are designed to monitor the activities of

host systems, such as mail servers, web servers, or individual workstations. NIDSs monitor the network traffic for multiple hosts by capturing and analyzing network packets. HIDSs are typically more expensive to deploy and manage, and may consume system resource, however they can detect insider attacks. NIDSs are easier to manage because they are platform independent and typically installed on a designated system, but they can only detect attacks which come through the network.

In general, intrusion detection methods are categorized into misuse and anomaly detection. In misuse detection, known attack patterns are stored in a database and then system activities are checked against these patterns. Such approach is also employed in commercial anti-virus products. Misuse detection systems may provide a high level of accuracy, but unable to detect novel attacks. In contrast, anomaly detection approaches learn normal system behavior and detect significant deviations from this baseline behavior. Anomaly detection systems (ADS) can detect novel attacks, but generate a prohibitive number of false alarms due in large part to the difficulty in obtaining complete descriptions of normal behavior.

Security events monitored for anomalies typically involve sequential and categorical records (e.g., audit trails, application logs, system calls, network requests) that reflect specific system activities and may indirectly reflect user behaviors. Hidden Markov Model (HMM) is a doubly stochastic process for sequential data Theoretical and empirical results have shown that, given an adequate number of hidden states and a sufficiently rich set of observations, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena (Bengio, 1999; Cappe et al., 2005; Ephraim and Merhav, 2002; Ghahramani, 2001; Poritz, 1988; Rabiner, 1989; Smyth et al., 1997). A well trained HMM provides a compact detector that captures the underlying structures of the monitored system based on the temporal order of events generated during *normal* operations. It then detects deviations from normal system behavior with high accuracy and tolerance to noise. ADSs based on HMMs have been successfully applied to model sequential security events occurring at different levels within a networking environment,

such as at the host level (Cho and Han, 2003; Hu, 2010; Lane and Brodley, 2003; Warrender et al., 1999; Yeung and Ding, 2003), network level (Gao et al., 2003; Ourston et al., 2003; Tosun, 2005), and wireless level (Cardenas et al., 2003; Konorski, 2005).

Traditional host-based anomaly detection systems monitor for significant deviation in operating system calls, as they provide a gateway between user and kernel mode (Forrest et al., 1996). In particular, abnormal behavior of privileged processes (e.g., root and setuid processes described in Section 1.2.1) is most dangerous, since these processes run with elevated administrative privileges. Flaws in privileged processes are often exploited to compromise system security. Various neural and statistical anomaly detectors have been applied within ADSs to learn the normal behavior of privileged processes through the system call sequences that are generated (Forrest et al., 2008; Warrender et al., 1999), and then detect deviations from normal behavior as anomalous. Among these, techniques based on discrete HMMs have been shown to produce a very high level of performance (Du et al., 2004; Florez-Larrahondo et al., 2005; Gao et al., 2002, 2003; Hoang and Hu, 2004; Hu, 2010; Wang et al., 2010, 2004; Warrender et al., 1999; Zhang et al., 2003).

This thesis focuses on HMM-based anomaly detection techniques for monitoring the deviations from normal behavior of privileged processes based on sequences of operating system calls.

Motivation and Problem Statement

Ideally, anomaly detection systems should efficiently detect and report all accidental or deliberate malicious activities – from insiders or outsiders, successful or unsuccessful, known or novel, without raising false alarms. In this ideal situation, the output of an ADS should also be readily usable, with no or limited user intervention. More importantly, an ADS must accommodate newly-acquired data and adapt to changes in normal system behavior over time, to maintain or improve its accuracy.

In practice, however, ADSs typically generate an excessive number of false alarms – a major obstacle limiting their deployment in real-world applications. A false alarm (or false positive) occurs when a normal event is misclassified as anomalous. False alarms cause expensive disruption due to the need to investigate, and ascertain or refute. It is often very difficult to determine the exact sequence of events that triggered an alarm. Furthermore, frequent false alarms reduce the confidence in the system, and lead operators to undermine the credibility of future alarms. This issue is compounded further when several IDSs are actively monitored by a single operator. In addition, when intrusion detection is coupled with active response capabilities[1] (Ghorbani et al., 2010; Rash et al., 2005; Stakhanova et al., 2007), frequent false alarms will affect the availability of system resources, because of possible service disruptions.

False alarms are caused by several reasons, including poorly designed detectors, unrepresentative normal data for training, and limited data for validation and testing. Poorly designed anomaly detectors can not precisely describe the typical behavior of the protected system, and hence normal events are misclassified as anomalous. Design issues typically include inadequate assumptions about the real system behavior, inappropriate model selection, and poor optimization of models parameters. For instance, the number of HMM states may have a significant impact on the detection accuracy. Overfitting is an important issue that leads to inaccurate predictions. It occurs when the training process leads to memorization of noise in training data, providing models that are overspecialized for the training data but perform poorly on unseen test data. In general, complex HMMs (with large number of states) trained on limited amount of data are more prone to overfitting.

Anomaly detectors based on HMMs require a representative amount of normal training data. In practice, a limited amount of representative normal data is typically provided for training, because the collection and analysis of training data is costly. The anomaly

[1] An IDS that actively responds to suspicious activities by, for instance, shutting systems down, logging out users, or blocking network connections is often referred to as an intrusion prevention systems (IPS) or an intrusion detection and prevention system (IDPS).

detector will therefore have an incomplete view of the normal process behavior, and hence misclassify rare normal events as anomalous. Furthermore, substantial changes to the monitored environment, reduce the reliability of the detector as the internal model of normal behavior diverges with respect to the underlying data distribution, producing both false positive and negative errors. Therefore, an ADS must be able to efficiently accommodate newly-acquired data, after it has originally been trained and deployed for operations, to maintain or improve system accuracy over time.

The incremental re-estimation of HMM parameters raises several challenges. Given newly-acquired data for training, HMM parameters should be updated from new data without requiring access to the previously-learned training data, and without corrupting previously-acquired knowledge (i.e., models of normal behavior) (Grossberg, 1988; Polikar et al., 2001). Standard techniques for training HMM parameters involve iterative batch learning, based either on the Baum-Welch (BW) algorithm (Baum et al., 1970), a specialized expectation maximization (EM) technique (Dempster et al., 1977), or on numerical optimization methods, such as the Gradient Descent (GD) algorithm (Levinson et al., 1983). These techniques assume a fixed (finite) amount of training data available throughout the training process. In either case, HMM parameters are estimated over several training iterations, until the likelihood function is maximized over data samples. Each training iteration requires applying the Forward-Backward (FB) algorithm (Baum, 1972; Baum et al., 1970; Chang and Hancock, 1966) on the *entire* training data to evaluate the likelihood value and estimate the state densities – the sufficient statistics for updating HMM parameters.

To accommodate newly-acquired data, batch learning techniques must restart HMM training from the start using all cumulative data, which is a resource intensive and time consuming task. Over time, an increasingly large space is required for storing cumulative training data, and HMM parameters estimation becomes prohibitively costly. In fact, considering the HMM parameters obtained from previously-learned data as starting point for training on newly-acquired data, may not allow for an optimization process that

escapes the local maximum associated with the previously-learned data. Being stuck in local maxima leads to knowledge corruption and hence to a decline in system performance. The time and memory complexity of BW or GD algorithm, for training an HMM with N states, is $\mathcal{O}(N^2T)$ and $\mathcal{O}(NT)$ respectively, for a sequence of length T symbols.

As an alternative, several on-line learning techniques have been proposed in literature to re-estimate HMM parameters from an infinite stream of observation symbols or sub-sequences. These algorithms re-estimate HMM parameters continuously upon observing each new observation symbol (Florez-Larrahondo et al., 2005; Garg and Warmuth, 2003; LeGland and Mevel, 1995, 1997; Mongillo and Deneve, 2008; Stiller and Radons, 1999) or new observation sub-sequence, (Baldi and Chauvin, 1994; Cappe et al., 1998; Mizuno et al., 2000; Ryden, 1997; Singer and Warmuth, 1996) typically without iteration. In practice however, on-line learning using blocks comprising a limited number of observation sub-sequences yields a low level of performance as one pass over each block is not sufficient to capture the phenomena.

A single classifier systems for incremental learning may approximate the underlying data distribution inadequately. Indeed, incremental learning using a single HMM with a fixed number of states may not capture a representative approximation of the normal process behavior. Different HMMs trained with different number of states capture different underlying structures of the training data. Furthermore, HMMs trained according to different random initializations may lead the algorithm to converge to different solutions in parameters space, due to the many local maxima of the likelihood function. Therefore, a single HMM not provide a high level of performance over the entire detection space.

Ensemble methods have been recently employed to overcome the limitations faced with a single classifier system (Dietterich, 2000; Kuncheva, 2004a; Polikar, 2006; Tulyakov et al., 2008). Theoretical and empirical evidence have shown that combining the outputs of several accurate and diverse classifiers is an effective technique for improving the overall system accuracy (Brown et al., 2005; Dietterich, 2000; Kittler, 1998; Kuncheva, 2004a;

Polikar, 2006; Rokach, 2010; Tulyakov et al., 2008). In general, designing an ensemble of classifiers (EoC) involves generating a diverse pool of base classifiers (Breiman, 1996; Freund and Schapire, 1996; Ho et al., 1994), selecting an accurate and diversified subset of classifiers (Tsoumakas et al., 2009), and then combining their output predictions (Kuncheva, 2004a; Tulyakov et al., 2008).

The combination of selected classifiers typically occurs at the score, rank or decision levels (Kuncheva, 2004a; Tulyakov et al., 2008). Fusion at the score level typically requires normalization of the scores before applying the fusion functions, such as sum, product, and average (Kittler, 1998). Fusion at the rank level is mostly suitable for multi-class classification problems where the correct class is expected to appear in the top of the ranked list (Ho et al., 1994; Van Erp and Schomaker, 2000). Fusion at the decision level exploits the least amount of information since only class labels are considered. Majority voting (Ruta and Gabrys, 2002), weighted majority voting (Ali and Pazzani, 1996; Littlestone and Warmuth, 1994) are among the most representative decision-level fusing functions. Combination of responses in the Receiver Operating Characteristic (ROC) space have been recently investigated as alternative decision-level fusion techniques. The ROC convex hull (ROCCH) or the maximum realizable ROC (MRROC) have been proposed to combine detectors based on a simple interpolation between their responses (Provost and Fawcett, 2001; Scott et al., 1998). Using Boolean conjunction or disjunction functions to combine the responses of multiple soft detectors in the ROC space have shown and improved performance over the MRROC (Haker et al., 2005; Tao and Veldhuis, 2008). However, these techniques assume conditionally independent classifiers and convexity of ROC curves (Haker et al., 2005; Tao and Veldhuis, 2008).

EoCs may be adapted for incremental learning by generating a new pool of classifiers as each block of data becomes available, and combining the outputs with those of previously-generated classifiers with some fusion technique (Kuncheva, 2004b). Most existing ensemble techniques proposed for incremental learning aim at maximizing the system performance through a single measure of accuracy (Chen and Chen, 2009; Muhlbaier et al.,

2004; Polikar et al., 2001). Furthermore, these techniques are mostly designed for two- or multi-class classification tasks, and hence require *labeled* training data sets to compute the errors committed by the base classifiers and update the weight distribution at each iteration. In HMM-based ADSs, the HMM detector is a one-class classifier that is trained using the normal system call data only. Moreover, an arbitrary and fixed threshold is typically set on the output scores of base classifiers prior to combining their decisions. This implicitly assumes fixed prior probabilities and misclassification costs. Nevertheless, the effectiveness of an EoC strongly depends on the decision fusion strategies. Standard techniques for fusion, such as voting, sum or averaging, assume independent classifiers and do not consider class priors (Kittler, 1998; Kuncheva, 2002). However, these assumptions are violated in anomaly detection applications, since detectors are typically designed using limited representative training data. Furthermore, the prior class distributions and misclassification costs are imbalanced, and may vary over time. In addition the correlation between classifiers decisions depends on the selection of decision thresholds.

Objectives and Contributions

This thesis addresses the challenges mentioned above and aims at improving the accuracy, efficiency and adaptability of existing host-based anomaly detection systems based on HMMs. *A major objective of this thesis is to develop adaptive techniques for ADSs based on HMMs that can maintain a high level of performance by efficiently adapting to newly-acquired data over time to account for rare normal events and adapt to legitimate changes in the monitored environments.*

To address the objectives mentioned above, this thesis presents effective techniques for adaptation of ADS based on HMMs. In response to new training data, these techniques allow for incremental learning at the decision and detection levels. At the decision level, this thesis presents a novel and general learn-and-combine approach based on Boolean combination of detector responses in the ROC space. It also proposes improved techniques for incremental learning of HMM parameters, which allow to accommodate new

data over time. In addition, this thesis attempts to provide an answer to the following question: *Given newly-acquired training data, is it best to adapt parameters of HMM detectors trained on previously-acquired data, or to train new HMMs on newly-acquired data and combine them with those trained on previously-acquired data?*

This thesis presents the following key contributions:

At the HMM Level.

- A survey of techniques found in literature that are suitable for incremental learning of HMM parameters. These techniques are classified according to the objective function, optimization technique, and target application, involving block-wise and symbol-wise learning of parameters. Convergence properties of these techniques are presented, along with an analysis of time and memory complexity. In addition, the challenges faced when these techniques are applied to incremental learning is assessed for scenarios in which the new training data is limited and abundant. As a result of this survey, improved strategies for incremental learning of HMM parameters are proposed. They are capable of maintaining or improving HMM accuracy over time, by limiting corruption of previously-acquired knowledge, while reducing training time and storage requirements. These incremental learning strategies provide efficient solutions for HMM-based anomaly detectors, and can be applied to other application domains.
- A novel variation of the Forward-Backward algorithm, called the Efficient Forward Filtering Backward Smoothing (EFFBS). EFFBS reduces the memory complexity, required for training an HMM with N states on a sequence of length T, from $\mathcal{O}(NT)$ to $\mathcal{O}(N)$, without increasing the computational cost. EFFBS is useful when abundant data (large T values) is provided for training HMM parameters.
- In depth investigation of the impact of several factors on HMM-based ADSs performance of training set size, number of HMM states, detector window size, anomaly size, and complexity and irregularity of the monitored process.

At the Decision Level.

- An efficient system is proposed for incremental learning of new blocks of training data using a learn-and-combine approach. When a new block of training data becomes available, a new pool of HMMs is generated using different number of HMM states and random initializations. The responses from the newly-trained HMMs are then combined to those of the previously-trained HMMs in ROC space using a novel incremental Boolean combination technique. The learn-and-combine approach allows to select a diversified ensemble of HMMs (EoHMMs) from the pool, and adapts the Boolean fusion functions and thresholds for improved performance, while it prunes redundant base HMMs.
- Since the pool size grows indefinitely as new blocks of data become available over time, employing specialized model management strategies is therefore a crucial aspect for maintaining the efficiency of an adaptive ADS. When a new pool of HMM is generated from a new block of data, the best EoHMMs is selected from the pool according to the proposed model selection algorithms (BC_{greedy} or BC_{search}), and then deployed for operations. Less accurate and redundant HMMs that have not been selected for some user-defined time interval are discarded from the pool.
  - The proposed Boolean combination techniques includes:
    - A new technique for Boolean combination (BC) of responses from two detectors in ROC space.
    - A new technique for cumulative Boolean combination of multiple detector (BCM) responses. BCM combines the results of BC (from the first two detectors) with the responses form the third detector, then with those of the fourth detector, and so on until the last detector. When applied to incremental learning from new data a variant of this algorithm is referred to as $incrBC$ to emphasize its incremental learning properties.
    - A new iterative Boolean combination (IBC) technique. IBC recombines original detector responses with the combination results of BCM over several iterations,

until the ROCCH stops improving (or a maximum number of iteration is performed).

- These techniques apply all Boolean functions to combine detector responses corresponding to each decision threshold (or each crisp detectors), and select the Boolean functions and decision thresholds (or crisp detectors) that most improve the overall ROCCH over that of original detectors. Since all Boolean functions are applied to explore the solution space, the proposed techniques require no prior assumptions about conditional independence of detectors or convexity of ROC curves, and their time complexity is linear with the number of soft detectors and quadratic with the number of crisp detectors. Improving the ROCCH minimizes both false positive and negative errors over the entire range of operating conditions, separately from assumptions about class distributions and error costs. More importantly, the overall composite ROCCH is always retained for adjusting the desired operating point during operations, and hence accounting for changes in prior probabilities and costs of errors.
- The proposed Boolean combination techniques are general in the sense that they can be employed to combine diverse responses of any set of crisp or soft one- or two-class classifiers, within a wide range of application domains. This includes combining the responses of the same detector trained on different data or features or trained according to different parameters, or from different detectors trained on the same data. In particular, they can be effectively applied when training data is limited and test data is imbalanced.

Proof-of-concept simulations are applied to adaptive anomaly detection from system call sequences. The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets[2]. They are conducted using a wide range of training set sizes with various alphabet sizes and complexities, and different anomaly sizes. The impact on performance of different techniques is assessed

[2] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

through different ROC measures, such as the area under the curve (AUC), partial AUC, and true positive rate (tpr) at a fixed false positive rate (fpr).

Organization of the Thesis

This manuscript-based thesis is structured into four chapters and five appendices. Figure 0.1, presents a schematic diagram of the thesis structure, and depicts the relationship between chapters and appendices. As illustrated in Figure 0.1 and further described next, most chapters (and appendices) consist of published (or submitted for publication) articles or papers in refereed scientific journals or conferences. The contents of each chapter is almost the same as that of the published paper, with minor modifications for consistency of notation throughout the thesis. Therefore, an overlap could not be avoided. Although each chapter may be read independently, it is recommended to read the thesis sequentially. However, readers unfamiliar with HMMs may benefit from reading Appendix I and II, prior to reading Chapter 2. Appendix V may also provide good introduction to Chapter 3. The rest of this thesis is structured as follows.

The first chapter provides an introduction to intrusion detection systems and their components, describes the general and basic concepts related to IDS, and presents a overview of relevant and related literature on host-based anomaly detection systems monitoring privileged process behavior by using system call sequences. It then focuses on process anomaly detection based on hidden Markov models. Both real and synthetic data sets employed in this research are also presented and the methodology considered for evaluation of IDSs performance is described. This chapter ends with the challenges facing process anomaly detection with HMMs.

Chapter 2 presents the survey of techniques found in literature that may be suitable for incremental learning of HMM parameters, which has been accepted in Information Sciences (Khreich et al., 2011a). Most techniques found in literature apply to on-line learning of HMM parameters from an infinite data stream. In appendix II, representative on-line techniques for training HMM parameters (from Chapter 2) are selected, extended

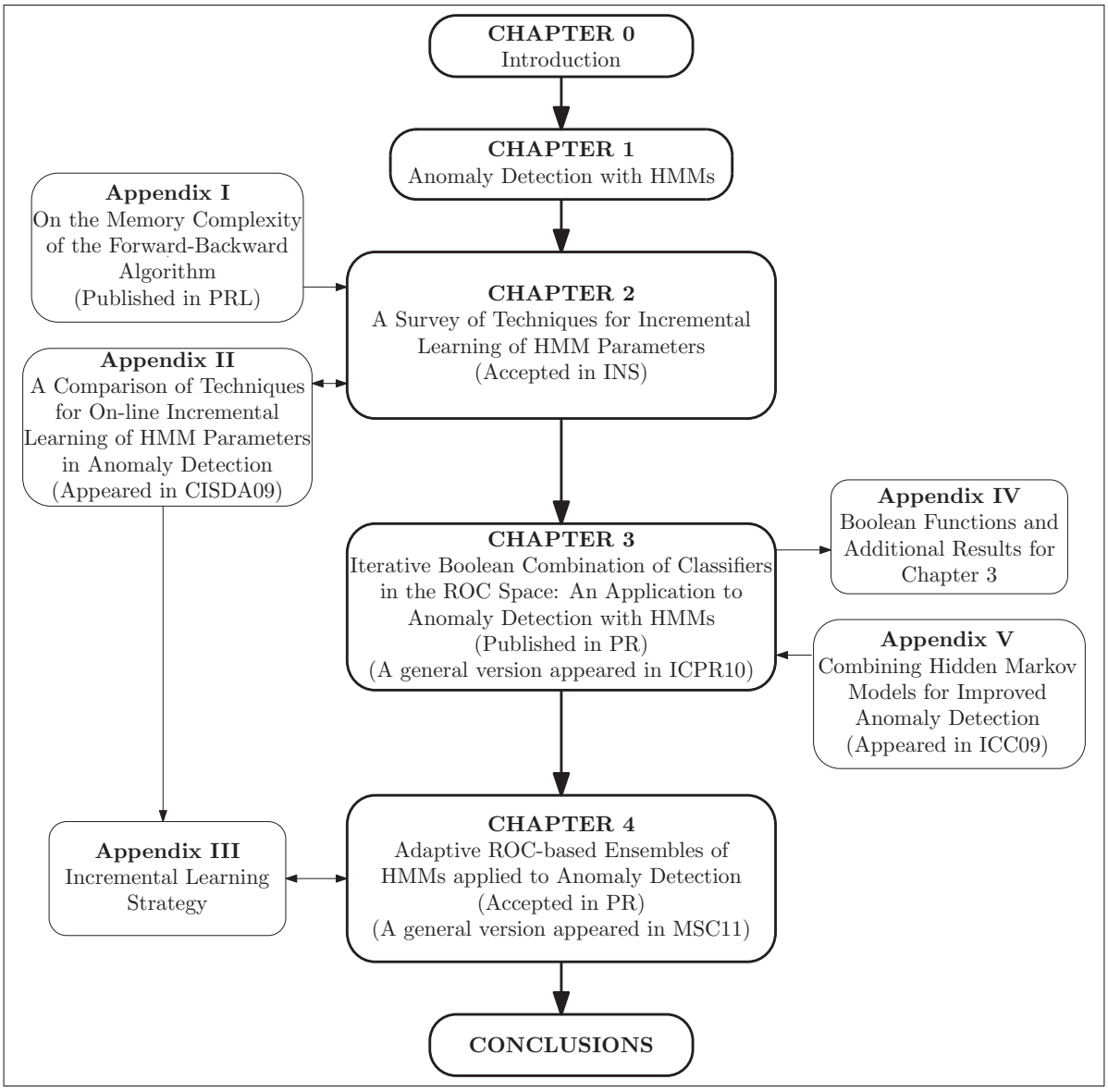


Figure 0.1 Structure of the thesis

to incremental learning, and compared to the original algorithms. Appendix II, appeared in the IEEE international conference on computational intelligence for security and defense applications (CISDA09) (Khreich et al., 2009a). Appendix III proposes strategies for further improvement of incremental learning of HMM parameters. A variation of these incremental learning strategies based on Baum-Welch algorithm (IBW) is applied in Chapter 4, and compared with the novel learn-and-combine approach proposed. Al-

though on-line learning techniques provide efficient solutions for training HMMs parameters from long observation sequences, they provide approximations to the smoothed state densities computed with the FB algorithm. The Efficient Forward Filtering Backward Smoothing (EFFBS) algorithm described in Appendix I, provides an exact alternative to the FB algorithm, with a memory complexity that is independent of the sequence length. This chapter has been published in Pattern Recognition Letters (Khreich et al., 2010c).

Chapter 3 describes the Boolean combination techniques proposed for efficient fusion of responses from detectors in the ROC space. The performance obtained by combining the responses of ADSs based on multiple HMMs, trained on *fixed-size* data sets, according to the Boolean combination techniques is compared to that of related techniques. This chapter has been published in Pattern Recognition (Khreich et al., 2010b). Further details about the Boolean functions and additional results are presented in Appendix IV. A version of this chapter, which addresses the Boolean combination in the more general decision-level fusion context, appeared in International Conference on Pattern Recognition (ICPR10) (Khreich et al., 2010a). The proposed Boolean combination techniques have been also successfully applied to combine different biometric systems for iris recognition (Gorodnichy et al., 2011). Appendix V presents an ADS with multiple-HMMs combined according to the MRROC technique. This chapter appeared in the International Conference on Communications (ICC09) (Khreich et al., 2009b).

Chapter 4 presents a novel anomaly detection system based on the learn-and-combine approach to accommodate newly-acquired system call data over time. It describes the generation, selection and pruning of HMMs, as well as selection and adaptation of Boolean functions and decision thresholds, when a new block of training data becomes available. The performance of the proposed system is compared to that of the reference batch BW (BBW), of on-line BW (OBW), and to the proposed incremental (IBW) described in Appendix III. This chapter has been accepted in Pattern Recognition (Khreich et al., 2011c). A version of this chapter, addressing incremental Boolean combination in the more general sense, has been also accepted in Multiple Classifier Systems (MSC11) (Khreich et al.,

2011b). Finally, a summary of contributions and discussion of key findings, followed by recommendations for future extensions to this research are presented in the conclusions.

CHAPTER 1

ANOMALY INTRUSION DETECTION SYSTEMS

This chapter presents an introduction to intrusion detection systems (IDSs). It starts with a general description of an IDS and its components, followed by an overview of common intrusion detection techniques in both host and network systems. It then focuses on host-based anomaly detection techniques using system call sequences generated by privileged processes. In particular, related work on HMM-based ADSs is presented and discussed in greater details. Both real and synthetic data sets employed in this research are presented next as well as the methodology considered for evaluation of IDSs performance. This chapter ends with a discussion on the challenges facing process anomaly detection with HMMs.

1.1 Overview of Intrusion Detection Systems

IDSs allow for the detection of successful or unsuccessful attempts to compromise systems security. An IDS is an important component of any security infrastructure that complements other security mechanisms. As illustrated in Figure 1.1, an IDS consists of four essential components: sensors, analysis engines, data repository, and management and reporting modules.

An IDS monitors the activity of a target system through a data source, such as system call traces, audit trails, or network packets. Relevant information from these data sources are captured by IDSs sensors, synthesized as events, and forwarded to the analysis engine for on-line analysis or to a repository for off-line analysis.

The analysis engine contains decision-making mechanisms to discriminate malicious events from normal events. It may include anomaly, misuse, or hybrid detection approaches (described next). Outputs from analysis engines include specific information regarding manifestation of suspicious events. These information are stored in a repository for foren-

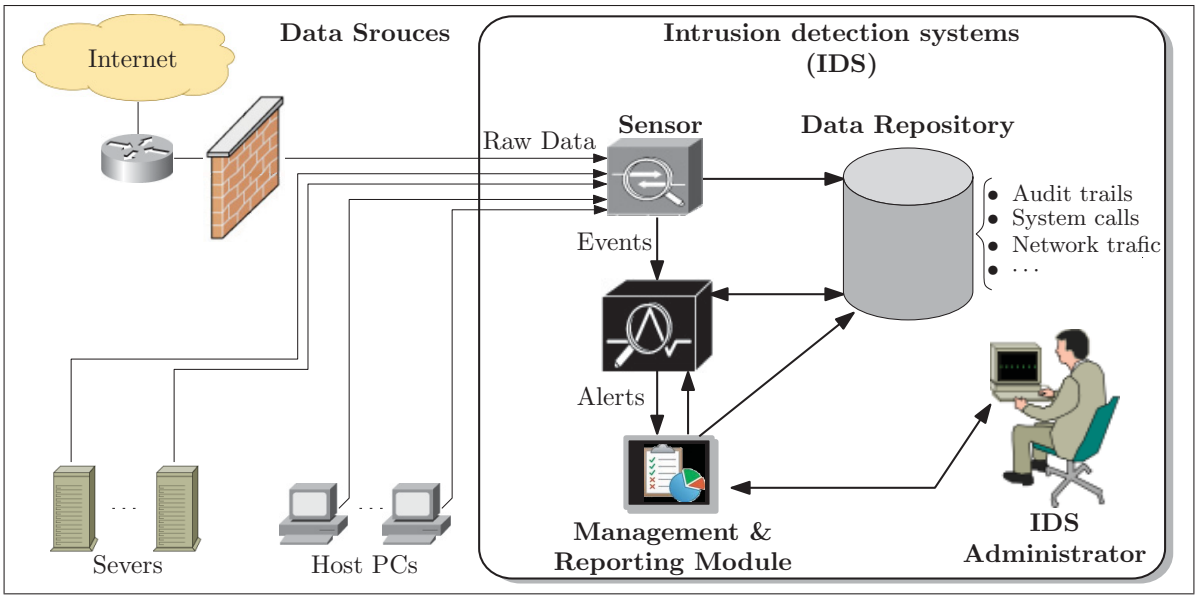


Figure 1.1 High level architecture of an intrusion detection system

sics analysis. A management and reporting module receives events that could indicate an attack from the analysis engine, raises an alarm to notify human operators, and reports the relevant information and the level of threat. The management module controls operations of IDS components, such as tuning decision thresholds of the analysis engine and updating the data repository.

The configuration of analysis engines, update of data repository, and response to alerts are among the responsibilities of the IDS administrator. When alerts are raised, the IDS administrator should prioritize and investigate incidents to refute or confirm that an attack has actually occurred. If an intrusion attempt is confirmed, a response team should react to limit the damage, and a forensic analysis team should investigate the cause of the successful attack. An IDS may include a response module that undertake further actions either to prevent an ongoing attack or to collect additional supporting information – it is often referred to as intrusion prevention system (IPS) or intrusion detection and prevention system (IDPS), (Ghorbani et al., 2010; Rash et al., 2005; Scarfone and Mell, 2007; Stakhanova et al., 2007).

IDSs are typically categorized depending on their monitoring scope (or location of the sensors) into network-based and host-based intrusion detection systems. They are also classified based on the detection methodology (employed by the analysis engine) into misuse and anomaly detection. More detailed taxonomies have been also developed, which further classify IDSs according to their architecture (centralized or fully distributed), behavior after attacks (passive or active), processing time (on-line or off-line), level of inspection (stateless or state full), etc. (Alessandri et al., 2001; Axelsson, 2000; Debar et al., 2000; Estevez-Tapiador et al., 2004; Lazarevic et al., 2005; Scarfone and Mell, 2007; Tucker et al., 2007).

1.1.1 Network-based IDS

Network-based IDSs (NIDSs) monitor the network traffic for multiple hosts by capturing and analyzing network packets for patterns of malicious activities. An NIDS is typically a stand-alone device that can control a network of arbitrary size with a small number of sensors (Lim and Jones, 2008). As illustrated in Figure 1.2, the sensors are often located at critical network junctures, such as network borders, demilitarized zone (DMZ)[1], or inside the local network (Northcutt and Novak, 2002; Proctor, 2000). NIDSs capture network traffic in promiscuous mode by connecting to a hub, network switch configured for port mirroring[2], or network tap[3]. Examples of open source NIDSs include Snort (Roesch, 1999) and Bro (Paxson, 1999).

NIDSs are platform independent, easy to deploy, and can cover large networks without consuming network or host resources. They are also able to detect unsuccessful attack attempts. However, an NIDS can detect only attacks which come through the monitored

[1]Demilitarized zone is a network segment located between a *secured* local network and unsecured external networks (Internet). DMZ usually contains servers that provide services to users on the external network, such as web, mail, and DNS servers, which must be hardened systems. Two firewalls are typically installed to form the DMZ.

[2]Port mirroring or spanning cross connects two or more ports on a network switch so that traffic can be simultaneously sent to a network analyzer connected to another port.

[3]A network tap is a direct connection between a sensor and the physical network. It provides the sensor with a copy of the data flowing across the network.

network links and has no visibility into individual systems, and hence unable to verify attack results. Advances in technology such as high-speed networks, switched networks and encrypted traffic have imposed several challenges on NIDSs. To inspect network traffic, a NIDS must reconstruct network data streams from every host. For instance, it must reassemble TCP/IP fragments and also take into account the variability in the implementation of these network protocols among different platforms. This causes a significant performance bottleneck and packet drop, especially with high-speed network throughputs. In addition, traffic reconstruction may open new doors for attackers (Peng et al., 2007; Ptacek and Newsham, 1998).

The migration from shared to switching environment has enhanced both network security and performance, but also complicates the allocation of NIDSs sensors. In traditional shared network, hubs were commonly used to connect segments of a local area network (LAN). When a packet arrives at one port, it is copied to the other ports to be available for all segments of the LAN. In switched networks messages are directly sent to the ultimate recipient, without broadcast and collision information over the network. Switching reduces the network traffic and makes it more difficult for intruders to sniff switched traffic, however it presents additional challenges on the distribution of sensors, especially if port spanning is not enabled on a switch. Enabling port spanning can also pose a risk if the spanned port is accessible to intruders.

In addition, network communications are increasingly using encryption technologies such as virtual private network (VPN), secure sockets layer (SSL), and secure shell (SSH) to conceal plain text. Encryption also prevents NIDSs from accessing and analyzing the content of network traffic. In fact, it is very difficult to analyze encrypted traffic until it has been decrypted with specific a application on the target host.

1.1.2 Host-based IDS

Host-based IDSs (HIDSs) are designed to monitor the activity of a host system, such as a mail server, web server, or an individual workstation. HIDSs identify intrusions by

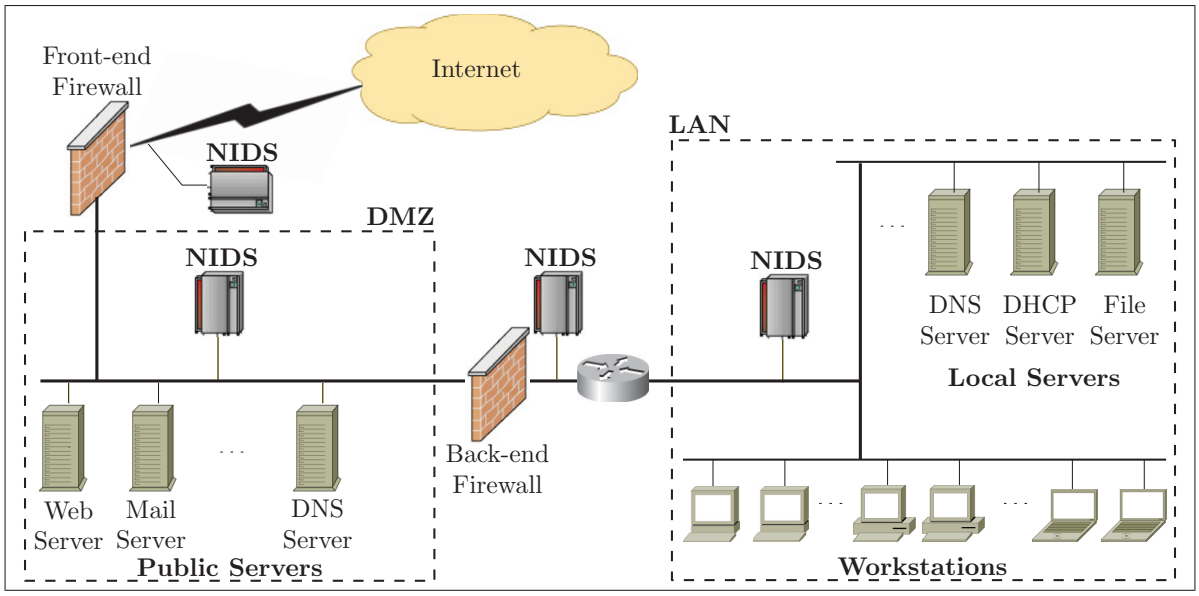


Figure 1.2 Host-based IDSs run on each server or host systems, while network-based IDS monitor network borders and DMZ

analyzing operating system calls, audit trails, application logs, file-system modifications (e.g., password files, access control lists, executable files), and other host activities and state (De Boer and Pels, 2005; Vigna and Kruegel, 2005). HIDSs are typically software-based systems which should be installed on every host of interest. In contrast with NIDSs, most HIDSs have software-based sensors that are able to extract sensitive (decrypted) information from a specific host at both kernel and user level, which make them more versatile systems. Examples of HIDSs include OSSEC (Bray et al., 2008) and NIDES (Anderson et al., 1995).

HIDSs are particularly effective at detecting insider attacks and verifying their success or failure. They are able to inspect low-level local system activities such as file access or modification time, changes to file permissions, or attempts to access privileged pro-

cesses or overwrite system executables; or to install new applications, Trojan horses[4] or backdoors[5], or other attacks that may involve software integrity breaches.

Although HIDSs typically require no dedicated hardware, they have to be tailored for a specific platform, and must be implemented on each individual machine. While this is manageable in a small homogeneous network, it may impose deployment and management difficulties as the network becomes larger and more heterogeneous. HIDSs are concerned only with individual systems and usually have limited view of network activity. Depending on the underlying detection methods, a HIDS can cause a significant degradation in hosts performance. In addition, HIDSs are prone to tampering since they are accessible to users.

1.1.3 Misuse Detection

Misuse detection approaches analyze host or network activity, looking for events that match patterns of known attacks (signatures). First a reference database of attack signatures is constructed, then monitored events from sensors data are compared against this database for evidence of intrusions. Signature matching is the most commonly employed misuse detection technique. For instance, Snort is a well-known open source signature-based network intrusion detection system (Roesch, 1999). Other misuse detection approaches include rule-based systems, state transition analysis, machine learning and data mining techniques.

In rule-based techniques, a set of *if-then-else* implication rules is used to characterize attacks (Lindqvist and Porras, 1999). Rule-based techniques are employed in various IDSs, such as EMERALD (Porras and Neumann, 1997) and NIDES (Anderson et al., 1995).

[4] A Trojan horse is a malicious program that masquerades as a legitimate application or file. Users are typically tricked into opening and executing it on their systems because it looks useful such as a music or picture file in E-mail attachments. Trojan infections range from annoying (modifying desktop appearance) to destructive (deleting files or changing information). Trojans are also commonly used to create backdoors.

[5] A backdoor is a tool installed on a compromised system to give an attacker direct access at a later date, without going through normal authentication or other security procedures. Typically, a backdoor program listens on specific ports to interact with the attackers.

State transition[6] techniques represent an attack as a minimal sequence of actions that an intruder must perform to break into a system, e.g., STAT (Ilgun et al., 1995). Recently, several machine learning and data mining techniques have been employed to extract attack signatures from collected data which contains some known attacks (Brugger et al., 2001; Helali, 2010; Lee et al., 2000).

Misuse detection systems provide a high level of detection accuracy without generating an overwhelming number of false alarms, but unable to detect novel attacks. Attacks for which signatures or rules have not been extracted yet, will go undetected. Signatures or rules describing new attacks must therefore be constantly updated. However, some time lag will inevitably occur before analyzing and incorporating patterns of newly deployed attacks, and thus novel attacks will always do some damage (zero-day attack).

In general signature definition is a difficult task. Loosely defined signatures decrease the detection accuracy and generate false alarms. However, by employing very specific signatures, a slight variation of the attack may not be detected (polymorphic attack). In practice, attackers attempt to deploy several variations of the same attack, and the vulnerabilities can change throughout the life cycle of a system and from one system to another. Therefore, the number of signatures or rules increases over time and may become unmanageable. Despite these limitations, misuse detection is still the most commonly applied approach in commercial IDS.

1.1.4 Anomaly Detection

Anomalies are patterns in data that do not conform to expected normal behavior. Anomaly detectors identify anomalous behavior on a host or network, under the assumption that attack patterns are typically different from those of normal behavior. An anomaly detection system (ADS) constructs a profile of expected normal behavior by using event data from users, processes, hosts,network connections, etc. These data sets

[6]States are used to characterize different snapshots of a system during the evolution of an attack, while transitions are user actions associated with specific events.

are typically collected over a period of normal (attack-free) operation to build normal profiles. During operation, an ADS attempts to detect sensor data events that deviate significantly from the normal profile. These deviations are considered as anomalous activities, however they are not necessarily malicious, since they may also correspond to rare normal events, program errors, or changes in normal behavior.

Several anomaly detection approaches have been proposed in literature (Chandola et al., 2009b), including statistical (Anderson et al., 1995; Markou and Singh, 2003a), rule-based (Vaccaro and Liepins, 1989), neural (Debar et al., 1992; Markou and Singh, 2003b), and machine learning approaches (Alanazi et al., 2010; Kumar et al., 2010; Ng, 2006; Tsai et al., 2009).

Unlike misuse detection techniques, anomaly detection is capable of detecting novel attacks. As discussed in Section , ADSs generate a large number of false alarms that would not be tolerated by the operator, and may decrease his confidence in the system. These alarms are not very instructive and require a costly and time-consuming investigation. In fact, ADSs can only detect symptoms of attacks without specific knowledge of details. However, anomaly detection remains an active intrusion detection research area.

1.2 Host-based Anomaly Detection

Host-based anomaly detection systems typically monitor the behavior of system processes[7] to determine whether a process is behaving normally or has been subverted by an attack. In particular, abnormal behavior of privileged processes is most dangerous. Attacks exploiting vulnerabilities in privileged processes can lead to full system compromise. In general, ADSs monitor events from variety of data sources, including log files created by a process and audit trails generated by the operating system. While these sources provide important security information, it is fairly easy to deploy attacks which do not leave any traces in traditional logs or audit trails, and hence evade detection. Sequences of system calls issued by a process to request kernel services, have been shown effec-

[7] A process is a program in execution.

tive in describing normal process behavior (Forrest et al., 1996). A substantial amount of research have investigated various techniques for detecting anomalies in system call sequences (Forrest et al., 2008; Warrender et al., 1999).

The remaining of this section provides a brief background on privileged processes and system calls and then reviews system call-based approaches to process anomaly detection.

1.2.1 Privileged Processes

A privilege is a permission to perform an action. The fundamental principles of least privilege and separation of privileges should be enforced on all subjects[8] at every level of the organization, for improved security, reliability, and accountability. According to the principle of least privilege (or need-to-know), all subjects should only be given enough privileges to perform their tasks (Saltzer and Schroeder, 1975). Ensuring least privilege requires identifying a subject tasks, determining the minimum set of privileges required to perform those tasks, and restricting the subject to a protection domain – a logical boundary that controls or prevents direct access among entities with different levels of privilege.

To enforce these principles, and control subjects' access to objects[9], modern computer systems provide several layers of protection domains. Modern processor architectures allow the operating system (OS) to run at different privilege levels (Silberschatz et al., 2008). In a typical dual-mode operation, user applications run in non-privileged *user mode*, while critical OS code runs in a privileged *kernel mode* and has access to system data and hardware. In user mode, programs have limited access to system information, and execute within their own virtual memory space.

In a multi-user operating system, two protection domains (user and kernel) are insufficient, since users need to share system objects and processes. For instance, in UNIX-like

[8] A subject is an active entity such as a user, administrator, process, or any other device that causes information to flow among objects or changes the system state.

[9] An object is a passive entity that contains or receives data, such as a data file, directory, printer, or other device.

OS, each user is associated with a protection domain. Each domain is a collection of access rights, each of which is an ordered a pair <object-name, rights-set>. When a user executes a program, the process runs in the protection domain associated with the user. Typically, two user identifiers (*uid*) are employed within the kernel. The *real uid* identifies the user who has created a process, and the *effective uid* allows a process to gain or drop privileges temporarily. Likewise, users with similar privileges have the same group identifier.

The *root*, or superuser, has the highest level of privilege ($uid = 0$); an absolute power over a system. Users are granted different levels of privileges by the root according to the organization's security policy. In some cases users privileges are insufficient to complete the required tasks. For example, if users are granted write access to the root-owned password file (`/etc/passwd`), they can alter other passwords; otherwise they cannot change their own password. UNIX-like OS employ the set user identifier (*setuid*) permission, which temporarily changes the *effective uid* of the process to that of the owner of the file, whereas the *real uid* is left unchanged. Therefore, the process inherits the privileges of the owner during the execution time of the required task, and then drops it. The *setuid* scheme allows low-privilege users to execute higher-privilege processes. A similar idea exists for groups.

Privileged processes and daemons[10] running with *setuid* permissions have been the source of endless security problems (Chen et al., 2002). Any process spawned by an elevated-privileges process, runs with these privileges. Buffer overflows are the most common attacks targeting privileged (root-owned) processes for escalation of privilege (AlephOne, 1996). Buffer overflow occurs when more data is written into the memory allocated to a variable than was allocated at compile time. Buffer overflow attacks exploit improper bounds checking on input data, to redirect program execution to arbitrary malicious code. If successful, the attacker may spawn a root shell and take full control over the

[10] A process continuously running in the background often with root or *setuid* permission and provides services from other processes.

system. Even though they have been reported for more than 15 years, buffer overflow vulnerabilities are likely to persist (Foster et al., 2005).

1.2.2 System Calls

System calls provide an interface between user and kernel mode. Users requests are made from applications or libraries[11] to the kernel via a platform-dependent set of system calls. A program that is not statically compiled will typically link to libraries at run-time to dynamically load required functions. Applications or libraries must invoke system calls to request kernel services, such as access to standard input/output, physical devices and network resources. Every system call has a unique number (known by the kernel) and a set of arguments. As illustrated in Figure 1.3, system calls are usually grouped according to their services. Examples of system calls for file management include `open()`, `read()`, `write()`, and `close()`. The length of a system call sequence generated by a process depends on the complexity and execution time of the process. The OS gains control upon a system call[12], switches to kernel mode, performs the requested service, and switches back to user mode (Silberschatz et al., 2008).

Manifestations of a wide variety of attacks, including buffer overflows, symbolic link, decode and SYN floods may appear at the system call level and differ from normal behavior of privileged processes, (Forrest et al., 1996; Hofmeyr et al., 1998; Kosoresow and Hofmeyer, 1997; Somayaji, 2002; Warrender et al., 1999). Furthermore, after gaining root privileges on a host, attackers will typically try to maintain administrative access on the compromised host (Blunden, 2009; Mitnick and Simon, 2005), by for instance installing backdoors and rootkits. Attackers will also attempt to distribute Trojan horses, using for instance comprised email addresses or shared network folders, to compromise other systems. These malicious activities may also invoke different sequences of system calls than those generated during normal execution of a process.

[11] In general, libraries runs in user mode. However, some libraries (e.g., standard C library) could offer a portion of system call interface.

[12] System calls require an architecture-specific control transfer, which is typically implemented through a software interrupt or trap. Interrupts transfer control form user mode to the kernel mode and back.

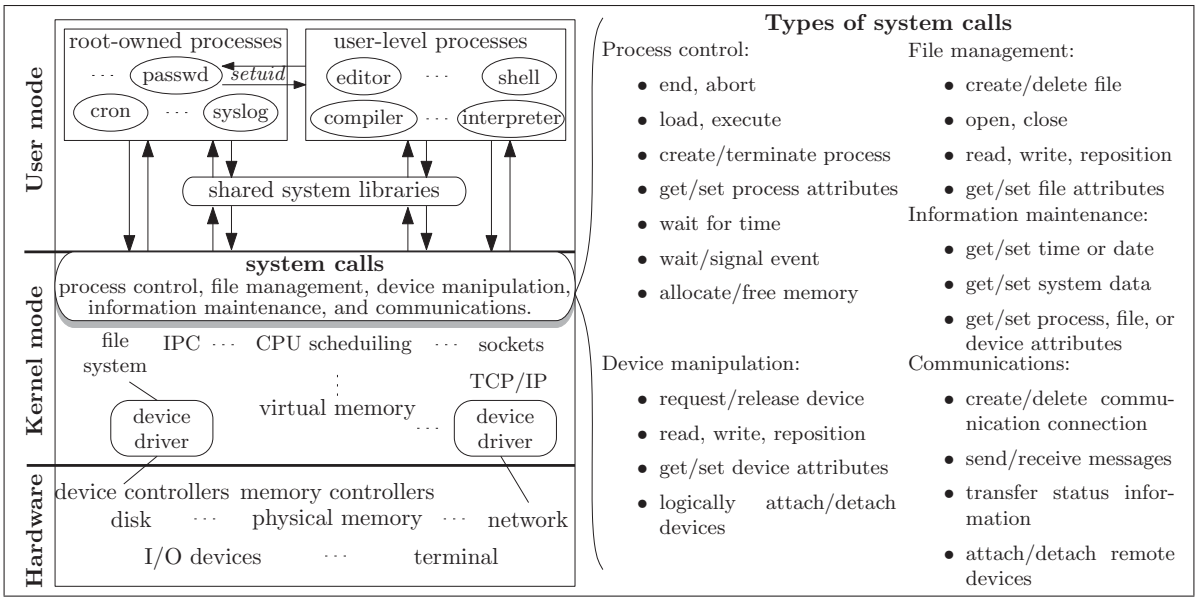


Figure 1.3 A high-level architecture of operating system, illustrating system-call interface to kernel and generic types of system calls

1.2.3 Anomaly Detection using System Calls

Forrest et al. (1996) were the first to suggest that the temporal order of system calls could be used to represent the normal behavior of a privileged process. They have collected system call data sets from various privileged process at the University of New Mexico (UNM) (as described in Section 1.4.1), and confirmed that short sub-sequences of system calls are consistent with normal process operation, and unusual burst will occur during an attack. Their anomaly detection system, called Time-Delay Embedding (TIDE), employed a sliding look-ahead window of a fixed length to record correlations between pairs of system calls. These correlations were stored in a database of normal patterns. During operations, the sliding window scheme is used to scan the system calls generated by the monitored process for anomalies – sub-sequences that are not found in the normal data. These anomalies were accumulated over the entire sequence and an alarm was raised if the anomaly count exceeded a user-defined threshold. Sequence time-delay embedding (STIDE) extended previous work by segmenting and enumerating system call sequences generated by a privileged process of interest into fixed-length

contiguous sub-sequences, using a fixed-size sliding window, shifted by one symbol (Warrender et al., 1999). During operations however, an anomaly score was defined as the number of mismatches in a temporally local region. To reduce the number of false alarms, a threshold was set for the anomaly score above which a sequence is considered as anomalous. Hamming-distance between sub-sequences has been also employed as a measure of anomalous behavior (Hofmeyr et al., 1998).

Several statistical and machine learning techniques, and other various extensions have been investigated over the last two decades for detecting system call anomalies using the UNM data sets Forrest et al. (2008). For instance, an inductive rule generator called RIPPER (Repeated Incremental Pruning to Produce Error Reduction) (Cohen, 1995), has been used for analyzing sequences of system calls and extracting rules (Fan et al., 2004). Finite state automata (FSA) have been proposed to model the system calls language, using deterministic or nondeterministic automatons (Michael and Ghosh, 2002; Sekar et al., 2001), or a call graph representation (Wagner and Dean, 2001). Lee and Xiang (2001) evaluated information-theoretic measures such as entropy and information cost. Application of machine learning techniques include neural network (Ghosh et al., 1999), k-nearest neighbors (Liao and Vemuri, 2002), n-grams Marceau (2000), Bayesian models (Kruegel et al., 2003), Markov models (Jha et al., 2001). Among these, techniques based on discrete HMMs have been shown to produce a high level of accuracy (Du et al., 2004; Florez-Larrahondo et al., 2005; Gao et al., 2002, 2003; Hoang and Hu, 2004; Hu, 2010; Wang et al., 2010, 2004; Warrender et al., 1999; Zhang et al., 2003). These HMM-based techniques are detailed in the next section.

1.3 Anomaly Detection with HMMs

Hidden Markov model is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, N, and a set of observation probability distributions, each one associated with a state. Starting from an initial state $S_i \in \{S_1, ..., S_N\}$, determined by the initial state probability distribution π_i,

at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij}. The process then emits a symbol v_k, from a finite alphabet $V = \{v_1, \ldots, v_M\}$ with M distinct observable symbols, according to the discrete-output probability distribution $b_j(v_k)$ of the current state S_j (Cappe et al., 2005; Elliott, 1994; Ephraim and Merhav, 2002; Rabiner, 1989).

Let $q_t \in S$ denotes the state of the process at time t, where $q_t = i$ indicates that the state is S_i at time t. An observation sequence of length T is denoted by $O = o_1, \ldots, o_T$ (or more concisely by $o_{1:T}$), where o_t is the observation at time t. An HMM is commonly parametrized by $\lambda = (\pi, A, B)$, where

- $\pi = \{\pi_i\}$ denotes the vector of initial state probability distribution,

$$\pi_i \triangleq P(q_1 = i)_{1 \leq i \leq N}$$

- $A = \{a_{ij}\}$ denotes the state transition probability distribution,

$$a_{ij} \triangleq P(q_{t+1} = j \mid q_t = i)_{1 \leq i,j \leq N}$$

- $B = \{b_j(k)\}$ denotes the state output probability distribution,

$$b_j(k) \triangleq P(o_t = v_k \mid q_t = j)_{1 \leq j \leq N, 1 \leq k \leq M}$$

The matrices A and B, and the transpose of vector π (π') are row stochastic, which impose the following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \forall i, \quad \sum_{k=1}^{M} b_j(k) = 1 \forall j, \quad \sum_{i=1}^{N} \pi_i = 1, \text{ and } a_{ij},\ b_j(k),\ \pi_i \in [0,1],\ \forall i, j, k$$

The theory underlying HMMs rely on the following assumptions:

- The first order Markov property (or limited horizon): The conditional probability distribution of current state depends only on previous state,

$$P(q_t = j \mid q_1, q_2, \ldots, q_{t-1}) = P(q_t = j \mid q_{t-1})$$

- Time-homogeneous (or time-invariant) assumption: The state transition probabilities are independent of the actual time at which the transitions occur,

$$P(q_t = j \mid q_{t-1}) = a_{ij}, \forall t$$

- Output independence assumption: The output generated at the current time step t depends solely on the current state q_t (independent of previous outputs),

$$P(o_t \mid q_1, q_2, \ldots, q_t) = P(o_t \mid q_t)$$

The success or failure of HMM application for pattern classification or recognition tasks rely on these fundamental assumptions. If the patterns considered can be described by stochastic processes without potentially violating these assumptions, then a well trained HMM may provide successful results. In fact, theoretical and empirical results have shown that, given an adequate number of states and a sufficiently rich set of data, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in terms of simple and compact models (Bengio, 1999; Bilmes, 2002). HMM has been successfully applied in various practical applications. It has become a predominant methodology for design of automatic speech recognition systems (Bahl et al., 1982; Huang and Hon, 2001; Rabiner, 1989). It has also been successfully applied to various other fields, such as communication and control (Elliott, 1994; Hovland and McCarragher, 1998), bioinformatics (Eddy, 1998; Krogh et al., 2001), computer vision (Brand and Kettnaker, 2000; Rittscher et al., 2000).

The output observations of HMMs can be discrete or continuous, scalars or vectors. As further detailed in Chapter 2, an HMM is termed discrete if the output alphabet is finite,

and continuous if the output alphabet is not necessarily finite, e.g., each state is governed by a parametric density function. Although, a continuous signal may be quantized into discrete observation, some applications, such as automatic speech recognition, attempt to avoid the inherent quantization error by using continuous output HMMs (Huang and Hon, 2001).

The transitions allowed between states and the number of states (or model order) determine the HMM topology. For instance, in ergodic (or fully connected) HMMs every state can be reached in a single step from any other state of the model. Strict left-right HMMs only allow self transition and transition from state S_n to S_{n+1}. An HMM that restricts transition from only left to right (but allow skipping states), is called Bakis HMM. In many real-world applications, the best topology is determined empirically. For example, in HMM application to automatic speech recognition, left-right topologies, where the state(s) represents a phoneme in a word, have been shown successful (Huang and Hon, 2001; Rabiner, 1989). In general, when the states have no physical meaning ergodic HMMs are employed.

In this thesis discrete-time finite-state HMMs are only considered, since they are representative of the intended application. In fact the process behavior can be described by finite set of hidden states, and there is only a finite alphabet of system calls. Furthermore, since the states have no physical representation, but only varied to best fit the considered process, ergodic HMMs are only considered.

In general, HMMs can perform the following canonical functions (Rabiner, 1989):

Evaluation aims to compute the likelihood of an observation sequence $o_{1:T}$ given a trained model λ, $P(o_{1:T} \mid \lambda)$. The likelihood is typically evaluated by using a fixed-interval smoothing algorithm such as the Forward-Backward (FB) (Rabiner, 1989) or the numerically more stable Forward-Filtering Backward-Smoothing (FFBS) (Ephraim and Merhav, 2002). See Appendix I for further details.

Decoding aims to find the most likely state sequence S that best explains the observation sequence $o_{1:T}$, given a trained HMM. The target is therefore to find the most likely state sequence S that maximizes $P(S \mid o_{1:T}, \lambda)$. The best state sequence is commonly determined by the Viterbi algorithm (Forney, 2005; Viterbi, 1967).

Learning aims to estimate the HMM parameters λ to best fit the observed data. Given a predefined topology and an observation sequence $o_{1:T}$, the target is to find the model λ that maximizes $P(o_{1:T} \mid \lambda)$. Unfortunately, there is no known analytical solution to the training problem. In practice, HMM parameters estimation is frequently performed according to the maximum likelihood estimation (MLE) criterion. MLE consists of maximizing the log-likelihood, $\log P(o_{1:T} \mid \lambda)$, of the training data over HMM parameters space. Unfortunately, since the log-likelihood depends on missing information (the latent states), there is no known analytical solution to the learning problem. In practice, iterative optimization techniques such as the Baum-Welch (BW) algorithm (Baum et al., 1970), a special case of the Expectation-Maximization (EM) (Dempster et al., 1977), or standard numerical optimization methods such as gradient descent (Baldi and Chauvin, 1994; Levinson et al., 1983) are often employed for this task. In either case, HMM parameters are estimated over several training iterations, until the likelihood function is maximized over data samples. Each training iteration typically involves observing all available training data to evaluate the log-likelihood value and estimate the state densities by using a fixed-interval smoothing algorithm.

Further details on the evaluation and parameter estimation of HMMs are presented in Chapter 2 and Appendix I. Decoding is less relevant to the application, since the hidden states have no physical. In fact, decoding the most likely state sequences (or uncovering the hidden states) is most commonly applied for recognition tasks, such as in automatic speech recognition (Bahl et al., 1982; Huang and Hon, 2001; Rabiner, 1989). For anomaly detection applications, HMMs (detectors) are typically trained on normal data and then employed to evaluate the test data, as described in the next section.

1.3.1 HMM-based Anomaly Detection using System Calls

In an effort to find the best technique for detecting anomalies in system call sequences generated by privileged processes, Warrender et al. (1999) have conducted a comparative benchmarking on UNM data sets (described in Section 1.4.1), using various techniques including sequence matching, data mining, and HMMs. The authors trained an ergodic HMM using BW algorithm on the normal system call sequences for each process in the UNM data sets. The number of HMM states (N) was selected heuristically. It is set roughly equal to the process alphabet size (Σ) – the number of unique system call symbols used by the process. For instance, they selected $N = 60$ states for sendmail process since its alphabet size $\Sigma = 53$ symbols. Each HMM is then tested on the entire anomalous sequences, looking for unusual state transitions or symbol outputs according to a *predefined* threshold. Their experimental results have shown that HMM-based anomaly detectors produce the highest level of detection accuracy on average, compared to other techniques at the expenses of expensive training resource requirements. Indeed, the time complexity of BW algorithm per iteration scales linearly with the sequence length and quadratically with the number of states. In addition, its memory complexity scales linearly with both sequence length and number of states (see Section 2.2 and Appendix I, for details).

Subsequent work addressed other variations of HMM training and testing techniques, as well as various alarm raising strategies, for detecting system calls anomaly using UNM data sets, in particular sendmail data sets. For instance, Wang et al. (2004) used BW algorithm to train ergodic HMMs with a fixed number of states ($N = 53$) on sendmail data, using different detector window sizes, DW, to assess the impact of DW on detection accuracy. The training sub-sequences are segmented from the original normal sequences using an overlapping sliding window of size DW. Anomalous sub-sequences are then labeled in comparison with normal the sub-sequences (of the same length) using STIDE. The accuracy is measured by counting the number of anomalies in a test sequence that are below a pre-specified threshold. Similarly, Du et al. (2004) trained HMM with fixed

number of states according to BW algorithm, and focused on the impact of DW on detection accuracy. However they introduced the notion of relative probability to raise alarms. This consists of computing the number of anomalous sub-sequences of length DW contained in a larger sliding window around the tested sub-sequence. An alarm is raised when the number of anomalies reaches an arbitrary threshold.

Gao et al. (2002) trained an ergodic HMM with $N = 60$ states using BW algorithm on the normal sub-sequences and a subset of the anomalous sub-sequences. Remaining subsets of anomalous sub-sequences, which are not included in training, are used for testing. The author also focused on the impact of detector window size and that of neighboring effects on detection accuracy. Zhang et al. (2003) proposed a detection method based on a hierarchical HMM to overcome the training computational time and memory costs. First, an HMM with a large number of states is trained on the entire data set, then HMM transition sequences are used to train another HMM with a lower number of states. The authors also applied this strategy to create profiles for both normal and anomalous data sets. These two approaches are based on misuse detection rather than an anomaly detection though.

Yeung and Ding (2003) compared ergodic and left-right HMM topologies using UNM data sets. STIDE was used for labeling training and testing sub-sequences. HMMs are trained according to BW algorithm on *unique* normal sub-sequences, using different values of sub-sequence length DW and number of states N. For each DW, they used the trained HMMs to compute the likelihood of all training sub-sequences and selected the minimal likelihood values as decision thresholds. Results have shown that ergodic HMMs outperform the left-right HMMs for a given number of states, and left-right HMMs have shown high sensitivity to the sequence length.

Qiao et al. (2002) trained ergodic HMMs according to BW algorithm using sendmail data sets and STIDE for labeling anomalous sub-sequences. However, a threshold is set on state transition only to discriminate between normal and intrusion behavior. A different

number of states have been used, with a fixed detector window size. The author noted the costly time and memory complexity with a large HMM state values.

Hoang and Hu (2004) proposed a combination technique at the detector level. It consists of training an HMM for each sub-sequence of observation symbols segmented from the entire sequence of observation, and then weight averaging the parameters of these HMMs to produce the combined model. The author claimed that this strategy may be used to incrementally combine ergodic HMMs. Although this technique may work for left-right HMMs, it is not suitable for ergodic HMMs as their states are permuted with each different training phase, and hence averaging parameters leads to knowledge corruption. This have been confirmed empirically in Section II.5 of Appendix II.

Florez-Larrahondo et al. (2005) proposed an on-line algorithm based on BW for efficient learning of HMM parameters from long sequence of observation (described in Section 2.3.2.1). The algorithm allows to learn to reduce the time and memory complexity in comparison to the traditional BW training. (Chen and Chen, 2009) employed this on-line algorithm to train five ergodic HMMs with the same number of states as base detectors in AdaBoost algorithm (Freund and Schapire, 1996) using fixed-sized data set. AdaBoost requires a labeled training set comprising normal and anomalous examples that can be weighted for importance. Therefore, the authors set an arbitrary threshold on the log-likelihood output from HMMs trained on the normal system call behavior, below which normal system call sequences are considered as anomalous. These HMMs are then updated according to an on-line AdaBoost variant (Oza and Russell, 2001) from new data. Threshold setting is shown to have a significant impact on the detection performance of the EoHMMs (Chen and Chen, 2009). In fact, rare system call events are normal, if they are considered anomalous during the design phase, they will generate false alarms during the testing phase. These rare events may be suspicious if, during operation, they occur in bursts over a short period of time.

A recent survey on HMM-based techniques for system call anomaly detection is provided by Wang et al. (2010).

1.4 Data Sets

1.4.1 University of New Mexico (UNM) Data Sets

The UNM data sets[13] are commonly used for benchmarking anomaly detections based on system calls sequences (Warrender et al., 1999). Normal system call sequences generated by privileged processes during (secured) normal operation were collected. These sequences are assumed attack-free and used for training the anomaly detectors. Different kinds of attacks such as Trojan and backdoor intrusion, buffer overflows, symbolic link, and decode attacks (Forrest et al., 1996; Hofmeyr et al., 1998; Kosoresow and Hofmeyer, 1997; Warrender et al., 1999) were launched against these processes, while collecting the system call sequences. These intrusive sequences comprise both normal and anomalous sub-sequences for testing, but specific labeling of these sub-sequences remains an issue.

In related work, intrusive sequences are usually labeled in comparison with normal sequences, using the STIDE matching technique. This labeling process considers STIDE responses as the ground truth, and leads to a biased evaluation and comparison of techniques, which depends on both training data size and detector window size. To confirm the results on system calls data from real processes, the same labeling strategy is used in this work. However fewer sequences are used to train the HMMs to alleviate the bias. Therefore, for each DW, STIDE is first trained on all available normal data from UNM sendmail (about 1.8 million system calls), and then used to label the corresponding sub-sequences ($AS = DW$) from the ten sequences available for testing (about $6,755$ system calls). The resulting labeled sub-sequences are concatenated, then divided into blocks of equal sizes, one for validation and the other for testing. During the experiments, smaller blocks of normal data are used for training the HMMs as normal system call observations are very redundant. In spite of labeling issues, redundant training data, and unrepresen-

[13] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

tative test data, UNM sendmail data set is the mostly used in literature due to limited publicly available system call data sets.

1.4.2 Synthetic Generator

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training and labeling) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations. The data generator is based on the Conditional Relative Entropy (CRE) of a source; it is closely related to the work of Florez-Larrahondo et al. (2005); Maxion and Tan (2000); Tan and Maxion (2002, 2003, 2005); Tan et al. (2002).

The CRE is defined as the conditional entropy divided by the maximum entropy (*MaxEnt*) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE is given by

$$CRE = \frac{-\sum_x p(x) \sum_y p(y \mid x) \log p(y \mid x)}{MaxEnt} \tag{1.1}$$

where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a sequence of system calls, the conditional probability, $p(y \mid x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $MM = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov model is first constructed by fixing some state transition values and varying other values, subject to $\sum_j a_{ij} = 1$, to obtain the desired CRE (see Figure 1.4). The resulting Markov matrix is then used as a generative model for normal data, as illustrated

$$MM_{(\Sigma=8,CRE=0.0)} = \begin{bmatrix} 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

$$MM_{(\Sigma=8,CRE=0.3)} = \begin{bmatrix} 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \\ 0.0003 & 0.0000 & 0.2365 & 0.2529 & 0.1364 & 0.2080 & 0.1450 & 0.0209 \\ 0.0003 & 0.0380 & 0.2630 & 0.2903 & 0.1865 & 0.1784 & 0.0435 & 0.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$$

$$MM_{(\Sigma=8,CRE=1.0)} = \begin{bmatrix} 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \\ 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 & 0.1250 \end{bmatrix}$$

Example of a sequence of length $T = 50$ observation symbols generated from $MM_{(\Sigma=8,CRE=0.0)}$:
8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1
Example of a sequence of length $T = 50$ observation symbols generated from $MM_{(\Sigma=8,CRE=0.3)}$:
7 6 5 6 5 6 4 5 6 6 7 2 3 4 5 6 3 4 5 6 6 7 3 4 5 6 3 4 5 6 3 4 5 6 3 4 5 6 3 4 5 6 6 3 4 5 6 4 5 6
Example of a sequence of length $T = 50$ observation symbols generated from $MM_{(\Sigma=8,CRE=1.0)}$:
8 7 1 6 3 6 1 7 6 6 3 3 7 4 6 2 6 3 4 5 6 8 2 8 4 2 6 3 3 8 6 1 3 2 1 6 1 3 2 1 8 4 4 2 2 7 6 4 5 1 2

Figure 1.4 Examples of Markov models with alphabet $\Sigma = 8$ symbols and various CRE values, each used to generate a sequence of length $T = 50$ observation symbols

in Figure 1.5. Sampling starts with a uniform initial state probability distribution. At each discrete time step, the process transits from state S_i to S_j according to the state probability distributions (a_{ij}), and outputs the value of new state (S_j), until the desired sequence length is reached.

The same Markov model, MM, is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly

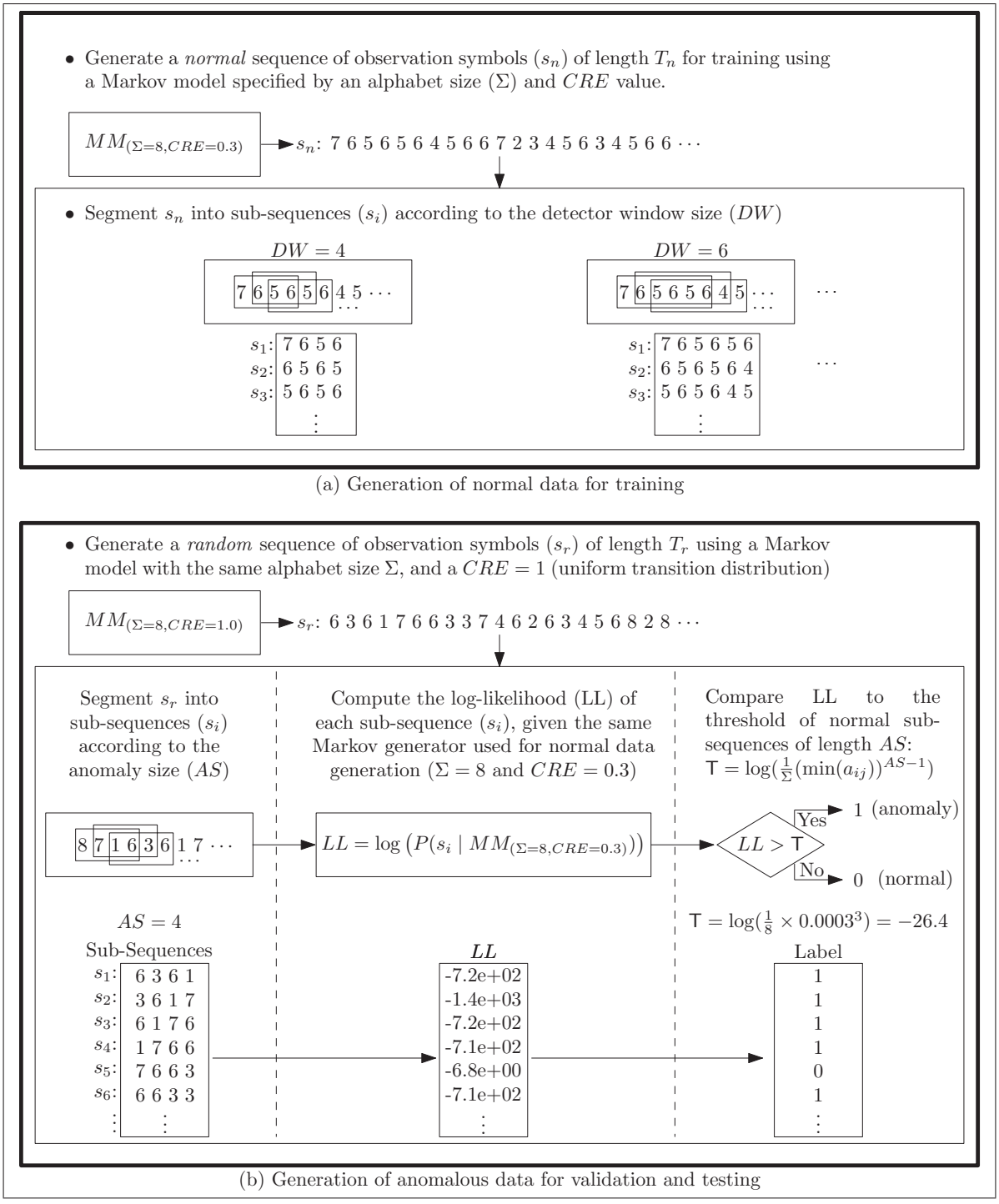


Figure 1.5 Synthetic normal and anomalous data generation according to a Markov model with $CRE = 0.3$ and alphabet of size $\Sigma = 8$ symbols

sequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly sequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly sequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test[14].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences (see Figure 1.4). As illustrated in Figure 1.5a, the normal data sequence with the desired length is then produced with the Markov model, and segmented using a sliding window (shift one) of a fixed size DW. To produce the anomalous data, a random sequence is generated using the same alphabet size Σ and a $CRE = 1$, which covers the entire normal and anomalous space. This random sequence is then segmented into sub-sequences of a desired length using a sliding window with a fixed size of AS (see Figure 1.5b). Then, the original generative Markov model is used to compute the likelihood of each sub-sequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $\mathsf{T} = \frac{1}{\Sigma}(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal positive value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol transitions in the sequence of size AS. This ensures that the anomalous sequences of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into this sequence at random according to a mixing ratio.

[14]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occur in high frequency over a short period of time.

1.5 Evaluation of Intrusion Detection Systems

The evaluation of intrusion detection systems is an open research topic, which face several challenges, including the lack of representative data and unified methodologies, and the employment of inadequate metrics for evaluation. (Abouzakhar and Manson, 2004; Gadelrab, 2008; Tucker et al., 2007). This thesis evaluates the proposed techniques for system call anomaly detection based on their efficiency, adaptability, and accuracy.

The efficiency considers the costs involved during the design and operation phase of an anomaly detection system. It considers the time and memory complexity required for designing an ADS, including detectors training, validation, selection or combination, as well as the space requirement for storing training data (e.g., batch technique) and for storing selected or combined models. Impact on efficiency of training set sizes with various alphabet sizes and complexities of monitored processes is also assessed. During operation, the efficiency considers the time and memory complexity required to operate the ADS, with one or multiple detectors, evaluate the likelihood of the input sub-sequences of observations, and make a decision.

A fully adaptive ADS must have mechanisms to detect legitimate changes in normal behaviors, collect data that reflect the changes, ensure the relevant data contain no pattern of attacks, and update its internal detectors and decision thresholds to adapt for the changes. The scope of this thesis is limited to adaptation at the detector and decision level. The remaining tasks are still under the administrator's scope of responsibility. Therefore, the adaptability of an ADS is evaluated for its effectiveness in maintaining or improving the overall system accuracy in response to new data. The accuracy of an ADS is determined based on the receiver operating characteristic analysis, and should not be confused with the accuracy measure (or inversely error rate), as described next.

1.5.1 Receiver Operating Characteristic (ROC) Analysis

This subsection provides relevant background details on ROC analysis, since it is heavily used in this research for evaluation and combination of detectors.

Given a detector and a test sample, there are four possible outcomes described by means of a confusion matrix, as illustrated in Figure 1.6. When a positive test sample (p) is presented to the detector and predicted as positive ($\hat{p}$) then it is counted as a true positive (TP); if it is however predicted as negative ($\hat{n}$) then it is counted as a false negative (FN). On the other hand, a negative test sample (n) that is predicted as negative ($\hat{n}$) is a true negative (TN), while it is a false positive (FP) if predicted as positive ($\hat{p}$). The true positive rate (tpr) is therefore the proportion of positives correctly classified (as positives) over the total number of positive samples in the test. The false positive rate (fpr) is the proportion of negatives incorrectly classified (as positives) over the total number of negative samples in the test. Similarly, the true negative rate (tnr) and false negative rate (fnr) can be defined over the negative class however.

A *crisp* detector outputs only a class label ($Y = 0$ or $Y = 1$), while a *soft* detector assigns scores or probabilities to the test samples ($\mathbf{x}$), by the means of a scoring function $f : \mathbf{x} \rightarrow \mathbb{R}$. Typically, the higher the score value, $f(\mathbf{x})$, the more likely the prediction of the positive event ($Y = 1$). A soft detector can be converted to a crisp one by setting a threshold T on the score values. A sample $\mathbf{x}$ is classified as positive ($Y = 1$) if it is assigned a score that is greater than or equal to T and negative ($Y = 0$) otherwise (see Figure 1.6).

A ROC curve is a two-dimensional curve in which the tpr is plotted against the fpr. An empirical ROC curve is typically obtained by connecting the observed (tpr, fpr) pairs of a detector for each decision threshold T. ROC curves can be efficiently generated by sorting the output scores of a detector, from the most likely to the least likely positive value, and considering unique values as decision thresholds (Fawcett, 2006). A crisp

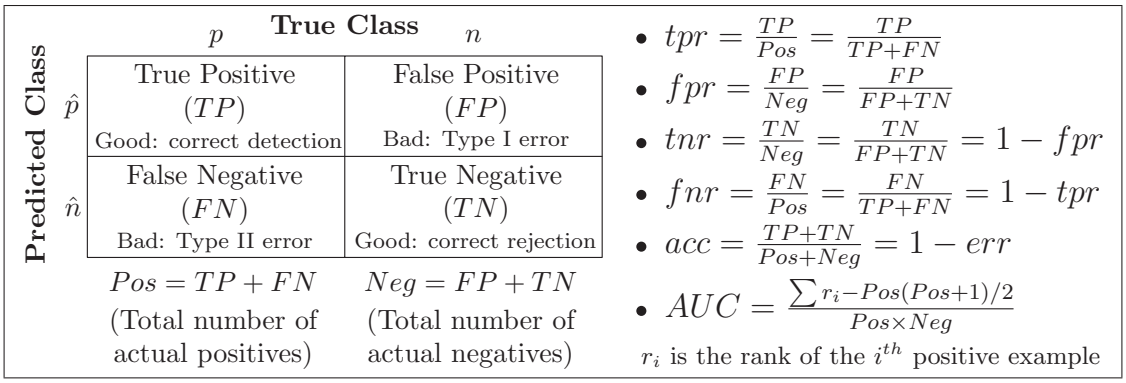


Figure 1.6 Confusion matrix and ROC common measures

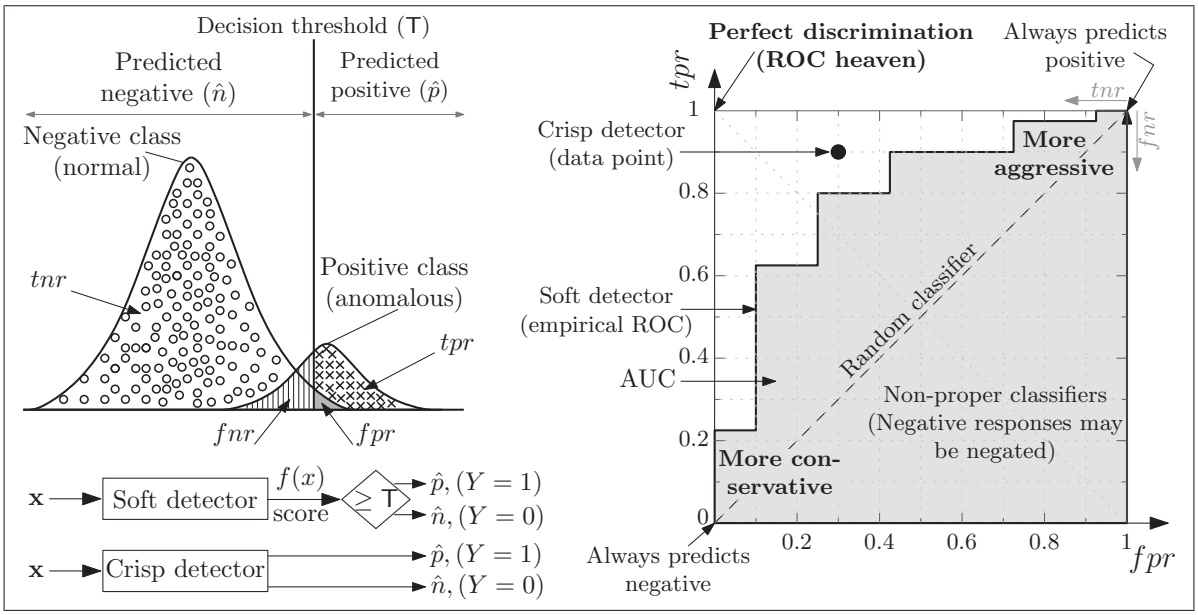


Figure 1.7 Illustration of the fixed *tpr* and fpr values produced by a crisp detector and the ROC curve generated by a soft detector (for various decision thresholds T). Important regions in the ROC space are annotated

detector however produces a single point (or data point) defined by its (fpr, *tpr*) in the ROC space (see Figure 1.7).

Given two operating points, say i and j, in the ROC space, i is defined as *superior* to j if $fpr_i \leq fpr_j$ and $tpr_i \geq tpr_j$. If one ROC curve I has all its points superior to those of another curve J, then I *dominates* J. This means that the detector of I always has a lower expected cost than that of J , over all possible class distributions and error costs .

If a ROC curve has $tpr_i > fpr_i$ for all its points i then, it is a *proper* ROC curve. ROC curves below the major diagonal can always be transformed into proper curves by using a linear transformation or changing the meaning of positive and negative.

ROC curves allows to visualize the performance of detectors and select optimal operational points, without committing to a single decision threshold. It presents detectors performance across the entire range of class distribution and error costs. For equal prior probability and cost of errors, the optimal decision threshold (minimizing overall errors and costs) corresponds to the vertex that is closest to the upper-left corner of the ROC plane (also called ROC heaven). In many practical cases, where prior knowledge of skewed class distributions or misclassification costs are available from the application domain, they should be considered while analyzing the ROC curve and selecting operational thresholds, using iso-performance lines (Fawcett, 2006; Provost and Fawcett, 2001).

For imprecise class distribution, the area under the ROC curve (AUC) has been proposed as more robust (global) measure for evaluation and selection of classifiers than accuracy (acc) – or conversely error rate (err), (Bradley, 1997; Huang and Ling, 2005; Provost et al., 1998). In fact, the acc depends on the decision threshold, and assume fixed distributions for both positive and negative classes. The AUC however is the average of the tpr over all values of the fpr (independently of decision thresholds and prior class distributions). It can be also interpreted in terms of class separation, as the fraction of positive–negative pairs that are ranked correctly, (see Figure 1.7). The AUC evaluates how well a classifier is able to sort its predictions according to the confidence it assigns to them. For instance, with an $AUC = 1$ all positives are ranked higher than negatives indicating a perfect discrimination between classes. A random classifier has an $AUC = 0.5$ that is both classes are ranked at random. For a crisp classifier, the AUC is simply the area under the trapezoid and is calculated as the average of the tpr and fpr values. For a soft classifier, the AUC may be estimated directly from the data either by summing up the areas of the underlying trapezoids (Fawcett, 2004) or by means of the Wilcoxon–Mann–Whitney (WMW) statistic (Hanley and McNeil, 1982).

In some cases ROC curves may cross, and hence detectors providing higher overall AUC values may perform *worse* than those providing lower AUC values, in a specific region of ROC space . In such cases, the partial area under the ROC curve (pAUC) (Walter, 2005), for instance the area under between $fpr = 0$ and $fpr = 0.1$ ($\text{AUC}_{0.1}$), could be useful for comparing the specific regions of interest (Zhang et al., 2002). If the AUC (or the pAUC) values are not significantly different, the shape of the curves might need to be looked at. It may also be useful to look at the tpr for a fixed fpr of particular interest. However, any attempt to summarize a ROC curve into a single number leads to information loss, such as errors and costs trade-offs, which reduces the system adaptability.

1.6 Anomaly Detection Challenges

1.6.1 Representative Data Assumption

Most work found in related literature (described in Section 1.2.3 and 1.3.1) considers static environments and assumes being provided with a sufficiently representative amount of clean (attack-free) system call data for training the anomaly detectors. Even under these ideal assumptions, anomaly detectors still face the following challenges.

Designing an HMM for anomaly detection involves estimating HMM parameters and the number of hidden states, N, from the training data. The value of N has a considerable impact not only on system accuracy but also on HMM training time and memory requirements. In the literature on HMMs applied to anomaly detection (Du et al., 2004; Gao et al., 2002; Hoang and Hu, 2004; Hu, 2010; Qiao et al., 2002; Warrender et al., 1999; Zhang et al., 2003), the number of states is often chosen heuristically or empirically using validation data. In fact, HMMs trained with different number of states are able to capture different underlying structures of data. Therefore, a single best HMM will not provide a high level of performance over the entire detection space (see Appendix V).

An open issue, for all window based anomaly detection, is the selection of the operating detector window size DW – the size of the sliding window used for testing. A small

DW value is always desirable since it allows faster detection and response, incurs less processing overhead , as described in Appendix V. However, choosing the smallest DW that has good discriminative abilities, depends on the anomaly size, AS, in the testing set. The anomaly size is not known a priori, may vary considerably in real world application, and could even be controlled by the attacker. In general, ADSs are designed to provide accurate results for a particular window size and anomaly size. Discriminative or sequence matching techniques, are only able to detect anomalies with sizes equal to that of the detector window ($AS = DW$). In addition, they provide blind regions in the detection space (Maxion and Tan, 2000; Tan and Maxion, 2002, 2003). These techniques will miss an anomalous sequence that are larger that the detector window size ($DW < AS$) if all of its sub-sequences (of size equal to that of DW) are normal. The detector window will slide on these normal sub-sequences, without being able to discover that the whole sequence is anomalous (Maxion and Tan, 2000; Tan and Maxion, 2002, 2003).

As a generative model, HMM is less sensitive to the detector window size and have no blind regions since it computes the likelihood of each observation sequence. Although not investigated in depth in this thesis, Chapter 3 and Appendix V, show that HMM detection ability increases with the detector window size, since the likelihood of anomalous sequences becomes smaller at a faster rate than normal ones, and hence easier to detect. Once designed and trained, an HMM provides a compact model that does not increase with the size of training data or detector window. In contrast, the dimensionality of the training sub-sequences space increases exponentially with DW, which makes discriminative techniques prone to the curse of dimensionality. In addition, matching techniques that are based on search procedure such as look-up tables in STIDE must compare inputs to all normal training sub-sequences. The number of comparisons also increases exponentially with the detector window size DW, while for HMM evaluation the time complexity grows linearly with DW .

To eliminate the dependence on DW, HMM is typically trained on the entire sequence of system call observations without segmentation (Lane, 2000; Warrender et al., 1999).

However, when learning from long sequences of observations the FB algorithm, employed within standard BW (Baum et al., 1970) or Gradient-based algorithms (Baldi and Chauvin, 1994; Levinson et al., 1983) for estimation of HMM parameters, may become prohibitively costly in terms of time and memory complexity (Lane, 2000; Warrender et al., 1999). This has been also noted in other fields, such as bioinformatics (Krogh et al., 1994; Meyer and Durbin, 2004) and robot navigation systems (Koenig and Simmons, 1996). For a sequence of length T and an ergodic HMM with N states, the memory complexity of the FB algorithm grows linearly with T and N, $\mathcal{O}(NT)$, and its time complexity grows quadratically with N and linearly with T, $\mathcal{O}(N^2T)$. The efficient forward filtering backward smoothing (EFFBS) algorithm proposed in Appendix I, provides an alternative to the FB algorithm, which a reduced memory complexity, $\mathcal{O}(N)$, i.e., independent of the sequence length T.

Another issue with related anomaly detection techniques resides in their evaluation methodology, in particular, the accuracy measure and arbitrary selection of decision thresholds. Most techniques consider the number of anomalous sub-sequences found in the test sequence as accuracy measure. A sub-sequence is considered as anomalous if its likelihood value assigned by the HMM is below a given threshold. The thresholds are typically selected according to the minimum likelihood value assigned by a trained HMM to the normal sub-sequences. Consequently, this evaluation methodology accounts only for one type of error, as it does not consider false negative errors. Furthermore, setting the decision threshold at the tail of the normal distribution does not provide the optimal (Bayes) decision, unless in trivial situations where the two distributions (normal and anomalous) are linearly separable (see Figure 1.7).

1.6.2 Unrepresentative Data

In practice however, unrepresentative data is typically provided for training. Acquiring a sufficient amount of clean data that represents the normal behavior of complex real-world processes is a challenging task. Although abundant data can be collected from a

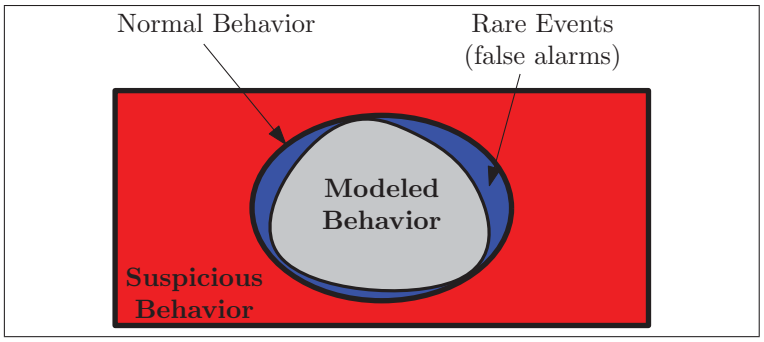


Figure 1.8 Illustration of normal process behavior when modeled with unrepresentative training data. Normal rare events will be considered as anomalous by the ADS, and hence trigger false alarms

constrained and overly secured environment,such as UNM data sets, this data still provide an incomplete view of the normal behavior of complex processes. This is due to the trade-off between process functionality and the restrictions required to achieve a high level of security, and hence "ascertain" that the collected data contains no attack patterns. More representative data can be collected from an open environment, in which a process is allowed to fully preform its intended functionality. However this data requires a time-consuming investigation by human specialists for the presence of attacks. Therefore, a limited amount of clean data is typically analyzed over time, and then provided for training the anomaly detector. The anomaly detector will also have an incomplete view of the normal process behavior. Incomplete view of the normal behavior leads to misclassifying rare normal events as anomalous, as illustrated in Figure 1.8.

Since HIDSs are deployed on each host, it is more practical to start from a generic model with limited view of normal behavior then update it, than to restart the whole training procedure on each host. For example, the same process of interest may have different settings and functionality on different hosts machines. It may be very costly to collect and analyze training data from the same process on every host and train a new model for each host. A less costly solution consists of training a generic HMM using normal data corresponding to common settings, an then incrementally update each model with

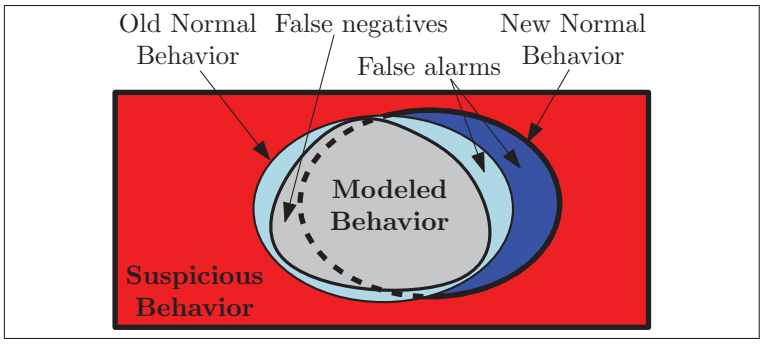


Figure 1.9 Illustration of changes in normal process behavior, due for instance to application update or changes in user behavior, and the resulting regions causing both false positive and negative errors

the required normal data to better fit the process normal behavior and hence reduce generation of false alarms.

Furthermore, the underlying data distribution of normal behavior may vary according to changes in the monitored environment after the ADS has been deployed for operations. For instance, an update to the code of the monitored process (e.g., to add newer functionality, fix some security holes, etc.) could lead the underlying distribution of normal behavior to drift. Normal behavior may also drift due to a change in the way users interact with the process. Changes in user behavior may occur when they are assigned new tasks or acquire new skills. Changes to the underlying normal distribution lead the model to deviate, and generate both false positive and negative error as illustrated in Figure 1.9. Accordingly, the model must be updated from new data to account from this change.

As a part of an ADS, the system administrator plays a crucial role in providing new data for training and validation of the detectors. When an alarm is raised, the suspicious system call sub-sequences are logged and analyzed for evidence of an attack. If an intrusion attempt is confirmed, the corresponding anomalous system calls should be provided to update the validation set. A response team should react to limit the damage, and a forensic analysis team should investigate the cause of the successful attack. Otherwise,

the intrusion attempt is considered as a false alarm and these rare sub-sequences are tagged as normal and employed to update the HMM detectors.

Therefore, an important feature of an ADS is the ability to accommodate newly-acquired data incrementally, after it has originally been trained and deployed for operations. Newly-acquired data allow to account for the incomplete view of normal process behavior and for possible changes that may occur over time. However, one major challenge is the efficient integration of the newly-acquired data into the ADS without corrupting the existing knowledge structure, and thereby degrading the performance.

CHAPTER 2

A SURVEY OF TECHNIQUES FOR INCREMENTAL LEARNING OF HMM PARAMETERS*

This chapter presents a survey of techniques found in literature that are suitable for incremental learning of HMM parameters. These techniques are classified according to the objective function, optimization technique, and target application, involving block-wise and symbol-wise learning of parameters. Convergence properties of these techniques are presented, along with an analysis of time and memory complexity. In addition, the challenges faced when these techniques are applied to incremental learning is assessed for scenarios in which the new training data is limited and abundant. While the convergence rate and resource requirements are critical factors when incremental learning is performed through one pass over abundant stream of data, effective stopping criteria and management of validation set are important when learning is performed through several iterations over limited data. In both cases managing the learning rate to integrate pre-existing knowledge and new data is crucial for maintaining a high level of performance. Finally, this chapter underscores the need for empirical benchmarking studies among techniques presented in literature, and proposes several evaluation criteria based on non-parametric statistical testing to facilitate the selection of techniques given a particular application domain.

*THIS CHAPTER IS ACCEPTED, UNDER REVISION, IN INFORMATION SCIENCES JOURNAL, ON MAY 30, 2011, SUBMISSION NUMBER: INS-D-10-141

2.1 Introduction

The Hidden Markov Model (HMM) is a stochastic model for sequential data. It is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, and a set of observation probability distributions, each one associated with a state. At each discrete time instant, the process is assumed to be in a state, and an observation is generated by the probability distribution corresponding to the current state. The HMM is termed *discrete* if the output alphabet is finite, and *continuous* if the output alphabet is not necessarily finite, e.g., each state is governed by a parametric density function (Elliott, 1994; Ephraim and Merhav, 2002; Rabiner, 1989).

Theoretical and empirical results have shown that, given an adequate number of states and a sufficiently rich set of data, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in terms of simple and compact models (Bengio, 1999; Bilmes, 2002). This is supported by the success of HMMs in various practical applications, where it has become a predominant methodology for design of automatic speech recognition systems (Bahl et al., 1982; Huang and Hon, 2001; Rabiner, 1989). It has also been successfully applied to various other fields, such as communication and control (Elliott, 1994; Hovland and McCarragher, 1998), bioinformatics (Eddy, 1998; Krogh et al., 2001), computer vision (Brand and Kettnaker, 2000; Rittscher et al., 2000), and computer and network security (Cho and Han, 2003; Lane and Brodley, 2003; Warrender et al., 1999). For instance, in the area of computer and network security, a growing number of HMM applications are found in intrusion detection systems (IDSs). HMMs have been applied either to anomaly detection, to model normal patterns of behavior, or in misuse detection, to model a predefined set of attacks. HMM applications in anomaly and misuse detection have emerged in both main categories of IDS - host-based IDS (Cho and Han, 2003; Lane and Brodley, 2003; Warrender et al., 1999; Yeung and Ding, 2003) and network-based IDS (Gao et al., 2003; Tosun, 2005). Moreover, HMMs have recently begun to emerge in wireless IDS applications (Cardenas et al., 2003; Konorski, 2005).

In many practical applications, the collection and analysis of training data is expensive and time consuming. As a consequence, data for training an HMM is often limited in practice, and may over time no longer be representative of the underlying data distribution. However, the performance of a generative model like the HMM depends heavily on the availability of an adequate amount of representative training data to estimate its parameters, and in some cases its topology. In static environments, where the underlying data distribution remains fixed, designing a HMM with a limited number of training observations may significantly degrade performance. This is also the case when new information emerges in dynamically-changing environments, where underlying data distribution varies or drifts in time. A HMM that is trained using data sampled from the environment will therefore incorporate some uncertainty with respect to the underlying data distribution (Domingos, 2000).

It is common to acquire additional training data from the environment at some point in time after a pattern classification system has originally been trained and deployed for operations. Since limited training data is typically employed in practice, and underlying data distribution are susceptible to change, a system based on HMMs should allow for adaptation in response to new training data from the operational environment or other sources (see Figure 2.1). The ability to efficiently adapt HMM parameters in response to newly-acquired training data, through *incremental learning*, is therefore an undisputed asset for sustaining a high level of performance. Indeed, refining a HMM to novelty encountered in the environment may reduce its uncertainty with respect to the underlying data distribution. However, incremental learning raises several issues. For one, HMM parameters should be updated from new data without requiring access to the previously-learned training data. In addition, parameters should be updated without corrupting previously-acquired knowledge (Polikar et al., 2001).

Standard techniques for estimating HMM parameters involve batch learning, based either on specialized Expectation-Maximization (EM) techniques (Dempster et al., 1977), such as the Baum-Welch algorithm (Baum et al., 1970), or on numerical optimization

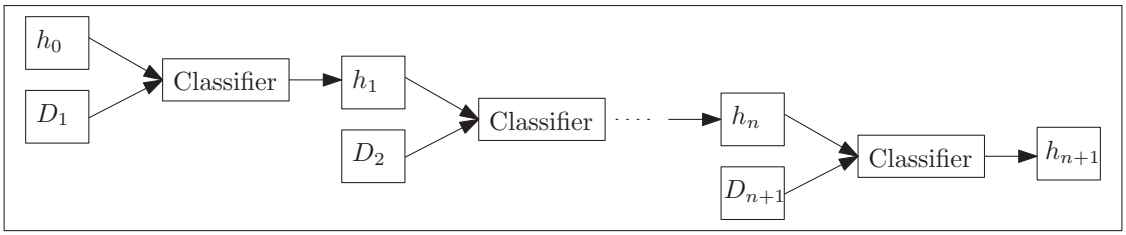


Figure 2.1 A generic incremental learning scenario where blocks of data are used to update the classifier in an incremental fashion over a period of time t. Let $D_1, \ldots, D_{n+1}$ be the blocks of training data available to the classifier at discrete instants in time $t_1, \ldots, t_{n+1}$. The classifier starts with initial hypothesis h_0 which constitutes the prior knowledge of the domain. Thus, h_0 gets updated to h_1 on the basis of D_1, and h_1 gets updated to h_2 on the basis of data D_2, and so forth (Caragea et al., 2001)

techniques, such as the Gradient Descent algorithm (Levinson et al., 1983). In either case, HMM parameters are estimated over several training iterations, until some objective function, e.g., maximum likelihood over some independent validation data, is maximized. For a batch learning technique, a finite-length sequence $O = o_1, o_2, \ldots, o_T$ of T training observations o_i is assumed to be available throughout the training process. Assuming that O is assembled into a block D of training data[1], each training iteration typically involves observing *all* subsequences in D prior to updating HMM parameters.

Given a new block D_2 of training data, a HMM that has previously been trained on D_1 through batch learning cannot accommodate D_2 without accumulating and storing all training data in memory, and training from the start using all of the cumulative data, $D_2 \bigcup D_1$. Otherwise, the previously-acquired knowledge may be corrupted, thereby compromising HMM performance. As illustrated in Figure 2.2, the HMM probabilities to be optimized may become trapped in local optima of the new cost function associated with $D_2 \bigcup D_1$. In fact, probabilities estimated after training on D_1 may not constitute a good starting point for training on D_2. Updating HMM parameters on all data using some batch learning technique may therefore incur a significant cost in terms of processing time and storage requirements. The time and memory complexity of standard techniques

[1]A block of training data is defined as a sequence of training observations that has been segmented into overlapping or non-overlapping subsequences according to a user-defined window size.

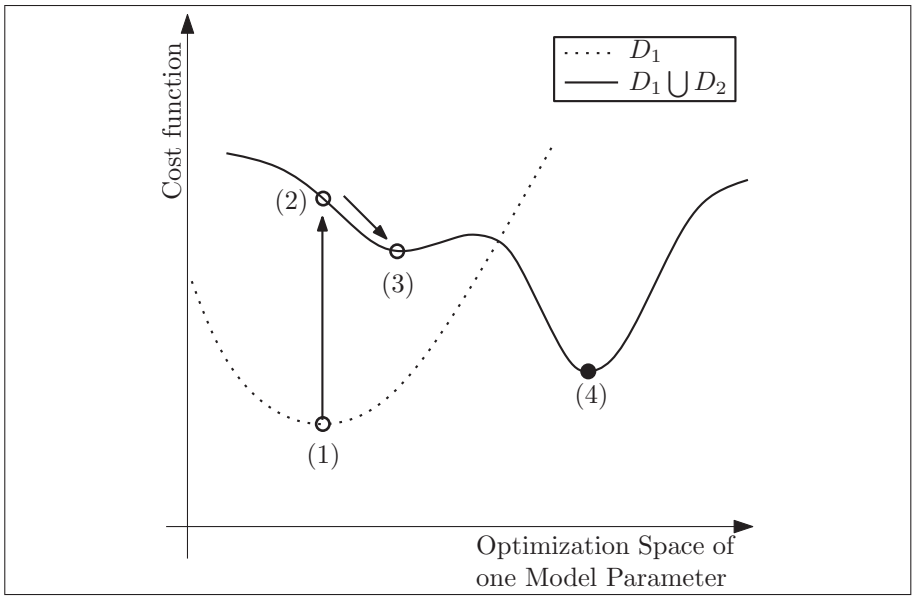


Figure 2.2 An illustration of the degeneration that may occur with batch learning of a new block of data. Suppose that the dotted curve represents the cost function associated with a system trained on block D_1, and that the plain curve represents the cost function associated with a system trained on the cumulative data $D_1 \bigcup D_2$. Point (1) represents the optimum solution of batch learning performed on D_1, while point (4) is the optimum solution for batch learning performed on $D_1 \bigcup D_2$. If point (1) is used as a starting point for incremental training on D_2 (point (2)), then it will become trapped in the local optimum at point (3)

grows linearly with the length, T, and number of training sequences R, and quadratically with the number of HMM states, N.

As an alternative, several on-line learning techniques proposed in literature may be applied for incremental learning. These include techniques based on EM, numerical optimization and recursive estimation, and assume the observation of stream of data. Some of these techniques are designed to update HMM parameters at a symbol level (symbol-wise), while others update parameters at a sequence level (block-wise). Techniques for on-line symbol-wise learning, also referred to as as recursive or sequential estimation techniques, are designed for situations in which training symbols are received one at a time, and HMM parameters are re-estimated upon observing each new symbol. Techniques for on-line block-wise learning are designed for situations in which training symbols are orga-

nized into a block of one or more sub-sequences, and HMM parameters are re-estimated upon observing each new sub-sequence of symbols. In either case, HMM parameters are updated from new training data, without requiring access to the previously-learned training data, and potentially without corrupting previously acquired knowledge.

The main advantages of applying these techniques to incremental learning are the ability to sustain a high level of performance, yet decrease the memory requirements, since there is no need for storing the data from previous training phases. Furthermore, since training is performed only on the new training sequences, and not on all accumulated data, on-line learning would also lower time complexity needed to learn new data. Finally, incremental learning may provide a powerful tool in a human-centric approach, where domain experts may be called upon to gradually design and update HMMs as the operational environment unfolds.

This chapter contains a survey of techniques that apply to incremental learning of HMM parameters[2]. These techniques are classified according to objective function, optimization technique, and target application that involve block-wise and symbol-wise learning of parameters. An analysis of their convergence properties and of their time and memory complexity is presented, and the applicability of these techniques is assessed for incremental learning scenarios in which new data is either abundant or limited. Finally, the advantages and shortcomings of these techniques are outlined, providing the key issues and guidelines for their application in different learning scenarios.

This chapter is structured according to five sections. The next section briefly reviews the batch learning techniques employed to estimate HMM parameters, and introduces the formalism needed to support subsequent sections. Section 2.3 provides a taxonomy of on-line learning techniques form literature that apply for incremental learning of HMM parameters. An analysis of their convergence properties and resource requirements is

[2]A predefined topology and permissible transitions between the states (e.g., ergodic or temporal) is assumed when learning HMM parameters. In many real-world applications, the best topology is determined empirically. Although adapting HMM topologies, or jointly HMM parameters and topologies, to new data may have a significant impact on performance, this issue is beyond the scope of the thesis.

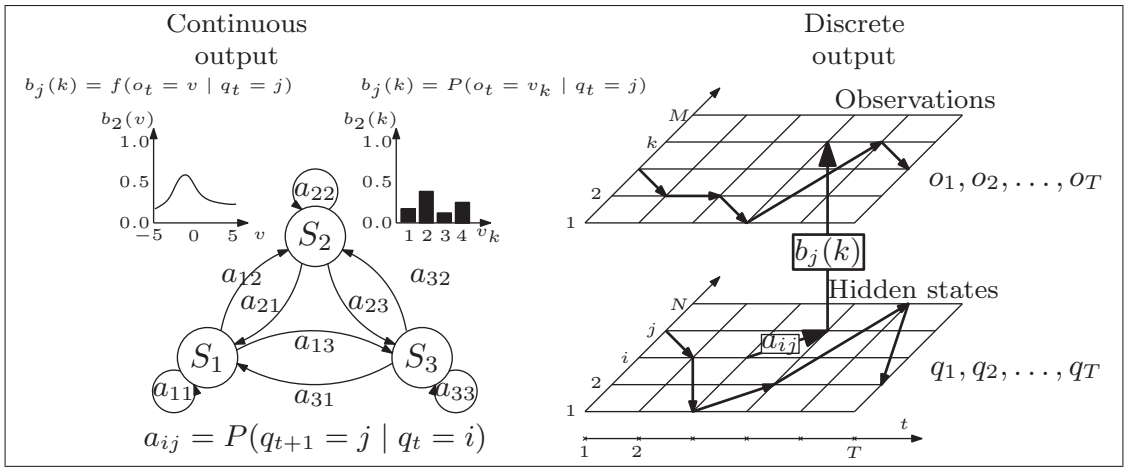


Figure 2.3 An illustration of an ergodic three states HMM with either continuous or discrete output observations (left). A discrete HMM with N states and M symbols transits between the hidden states q_t, and generates the observations o_t (right)

provided in Section 2.4. Then, an analysis of their potential applicability in different incremental learning scenarios is presented in Section 2.5. This chapter concludes with a discussion of the main challenges to be addressed for incremental learning of HMM parameters.

2.2 Batch Learning of HMM Parameters

A discrete-time finite-state HMM consists of N hidden states in the finite-state space $S = \{S_1, ..., S_N\}$ of the Markov process. Starting from an initial state S_i, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij}. The process then emits a symbol v according to the output probability distribution $b_j(v)$ of the current state S_j (see Figure 2.3). With a discrete HMM, the output $b_j(v)$ is finite, and with a continuous HMM, the output alphabet is governed by a parametric density function.

Let $q_t \in S$ denotes the state of the process at time t, where $q_t = i$ indicates that the state is S_i at time t. An observation sequence of length T is denoted by $O = o_1, \ldots, o_T$, where

o_t is the observation at time t. The sub-sequence $o_m, o_{m+1}, \ldots, o_n$, $n > m$, by the concise notation $o_{m:n}$. The HMM is then usually parametrized by $\lambda = (\pi, A, B)$, where

- $\pi = \{\pi_i\}$ denotes the vector of initial state probability distribution, $\pi_i \triangleq P(q_1 = i)_{1 \leq i \leq N}$
- $A = \{a_{ij}\}$ denotes the state transition probability distribution, $a_{ij} \triangleq P(q_{t+1} = j \mid q_t = i)_{1 \leq i,j \leq N}$
- $B = \{b_j(k)\}$ denotes the state output probability distribution,

  Aligned symbols with upper level text .

  - $b_j(k) \triangleq P(o_t = v_k \mid q_t = j)_{1 \leq j \leq N, 1 \leq k \leq M}$ for a finite and discrete alphabet $V = \{v_1, v_2, \ldots, v_M\}$, $v_k \in \mathcal{R}^L$ with M distinct observable symbols.
  - $b_j(v) \triangleq f(o_t = v \mid q_t = j)_{1 \leq j \leq N}$ for an infinite and continuous alphabet $V = \{v \mid v \in \mathcal{R}^L\}$. For instance, if the observation density for each state in the HMM is described by a univariate Gaussian distribution[3] $b_j(o_t) \sim \mathcal{N}(\mu_j, \sigma_j)$, for a scalar observation o_t, with a mean μ and a variance σ^2 then the state output density is given by:

$$b_j(o_t \mid \mu_j, \sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j}} \exp\left[-\frac{(o_t - \mu_j)^2}{2\sigma_j^2}\right] \tag{2.1}$$

For a finite-discrete HMM, both A, B and π' (the transpose of vector π) are row stochastic, which impose the following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \forall i, \sum_{k=1}^{M} b_j(k) = 1 \forall j, \text{ and } \sum_{i=1}^{N} \pi_i = 1 \tag{2.2}$$

$$a_{ij}, b_j(k), \pi_i \in [0, 1], \forall ijk \tag{2.3}$$

[3]It is usually termed as finite-state Markov chains in white Gaussian noise in control and communication community. It is also referred to as normal HMM.

For a continuous HMM, other constraints arise depending on the probability density function (pdf) of the states. For example, in the Gaussian distribution case, (2.1), one must ensure that the standard deviation is always positive, $\sigma_j > 0, \forall j$.

Given a HMM initialized according to the constraints described so far, there are three tasks of interest – the evaluation, decoding, and training tasks (Rabiner, 1989). The rest of this section focuses on batch learning techniques applied to address the third task, and in particular estimating HMM parameters, along with the definitions needed for future sections. For further details regarding the HMM, the reader is referred to the extensive literature (Elliott, 1994; Ephraim and Merhav, 2002; Rabiner, 1989).

Standard techniques for estimating HMM parameters $\lambda = (\pi, A, B)$ involve *batch learning* are based either on expectation-maximization or numerical optimization techniques. With batch learning, a finite-length sequence $O = o_1, o_2, ..., o_T$ of T training observations o_i is assumed to be available throughout the training process. The parameters are estimated over several training iterations, until some objective function is maximized. Each training iteration involves observing all the observation symbols prior to updating HMM parameters.

2.2.1 Objective Functions:

The estimation of HMM parameters is frequently performed according to the Maximum Likelihood Estimation (MLE) criterion. Other criteria such as the maximum mutual information (MMI), and minimum discrimination information (MDI) also be used for estimating HMM parameters (Ephraim and Rabiner, 1990). However, the widespread use of the MLE for HMM is a result of its attractive statistical properties – *consistency* and *asymptotic normality* – proven under quite general conditions (Bickel et al., 1998; Leroux, 1992). MLE consists in maximizing the likelihood or equivalently the log-likelihood of

the training data with regard to the model parameters:

$$\ell_T(\lambda) \triangleq \log P(o_{1:T} \mid \lambda) = \sum_S \log P(o_{1:T}, S \mid \lambda) = \sum_S \log P(o_{1:T} \mid S, \lambda) P(S \mid \lambda) \tag{2.4}$$

over the model parameter space Λ:

$$\lambda^* = \arg\max_{\lambda \in \Lambda} \ell_T(\lambda) \tag{2.5}$$

There is no known analytical solution to HMM parameters estimation since the log-likelihood function (2.4) depends on the unknown probability values of the latent states, S. In practice, iterative optimization procedures such as the expectation-maximization or the standard numerical optimizations techniques, which are described in the following sub-sections, are usually employed. In order to proceed iteratively any numerical optimization procedure must also evaluate the log-likelihood function at any value. However, a direct evaluation of the log-likelihood function (2.4) requires a summation over all hidden state paths $q_{1:T} \in S$, which has a prohibitively costly time complexity $\mathcal{O}(TN^T)$. Fortunately, there exists efficient recursive procedures for evaluating the log-likelihood values as well as estimating the conditional state (2.6) and joint state (2.7) densities associated with a given observation sequence $o_{1:t}$:

$$\gamma_{\tau|t}(i) \triangleq P(q_\tau = i \mid o_{1:t}, \lambda) \tag{2.6}$$

$$\xi_{\tau|t}(i,j) \triangleq P(q_\tau = i, q_{\tau+1} = j \mid o_{1:t}, \lambda) \tag{2.7}$$

in order to provide an optimal estimate, in the minimum mean square error (MMSE) sense, for the unknown state $\hat{q}_{\tau|t}(i)$ frequency – the key problem for HMM parameters estimation:

$$\hat{q}_{\tau|t}(i) = E\{q_\tau = i \mid o_{1:t}\} = \sum_{i=1}^{N} \gamma_{\tau|t}(i) \tag{2.8}$$

In estimation theory, this conditional estimation problem is called *filtering* if $\tau = t$; prediction if $\tau < t$, and smoothing if $\tau > t$. The smoothing problem is termed *fixed-point* smoothing when computing the $E\{q_\tau \mid o_{1:t}\}$ for a fixed τ and increasing $t = \tau, \tau+1, \ldots$, *fixed-lag* smoothing when computing the $E\{q_\tau \mid o_{1:t+\Delta}\}$ for a fixed lag $\Delta > 0$, and *fixed-interval* smoothing when computing the $E\{q_\tau \mid o_{1:T}\}$ for all $\tau = 1, 2, \ldots t, \ldots, T$.

The fixed-interval smoothing problem is therefore to find the best estimate of the states at any time conditioned on the entire observations sequence, which is typically performed in batch learning using the Forward-Backward (FB) (Baum, 1972; Baum et al., 1970; Chang and Hancock, 1966) or the Forward-Filtering Backward-Smoothing (FFBS) (Cappe and Moulines, 2005; Ephraim and Merhav, 2002) algorithms. Typically, fixed-interval smoothing algorithms involve an estimation of the filtered state density, for $t = 1, \ldots, T$:

$$\gamma_{t|t}(i) \triangleq P(q_t = i \mid o_{1:t}, \lambda) \tag{2.9}$$

which provides the best estimate of the conditional distribution of states given the past and present observations. It can be computed from the predictive state density (2.10), which provides the best estimate of the conditional distribution of states given only the past observations,

$$\gamma_{t|t-1}(i) \triangleq P(q_t = i \mid o_{1:t-1}, \lambda);\ \gamma_{1|0}(i) = \pi_i \tag{2.10}$$

according to the following recursion:

$$\gamma_{t|t}(i) = \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{j=1}^{N} \gamma_{t|t-1}(j) b_j(o_t)} \tag{2.11}$$

The predictive state density (2.10) at time $t+1$ can be then computed from the filtered state density (2.9) at time t:

$$\gamma_{t+1|t}(j) = \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij} \tag{2.12}$$

Given an observation sequence $o_{1:T}$, the log-likelihood is therefore evaluated with a time complexity of $\mathcal{O}(N^2T)$:

$$\ell_T(\lambda) = \sum_{t=1}^{T} \log P(o_t \mid o_{1:t-1}) = \sum_{t=1}^{T} \log \sum_{i=1}^{N} b_i(o_t)\gamma_{t|t-1}(i) \tag{2.13}$$

The predictive state density (2.10) is in fact a fixed-lag smoothing with one symbol look-ahead ($\Delta = 1$). In situations where some delay or latency between receiving the observations and updating HMM parameters can be tolerated, incorporating more lag provides stability and improved performance over filtering estimation (Anderson, 1999). This is because the latter relies only on the information that is available at the current time. Although they provide an approximation of the conditional state distributions, both filtering and fixed-lag smoothing have been the core of several on-line learning techniques, presented in Section 2.3, since their recursions can go on indefinitely in time providing the state estimates without (or with a small fixed) delay. In addition, they require a linear memory complexity $\mathcal{O}(N)$ that is independent of the observation length T, while maintaining a low computational complexity $\mathcal{O}(N^2T)$.

The backward recursion of the FFBS (or FB) algorithm exploits the filtered state estimates in order to provide an exact smoothed estimate for the states, for $t = T-1, \ldots, 1$:

$$\gamma_{t|T}(i) = \sum_{j=1}^{N} \xi_{t|T}(i,j) \tag{2.14}$$

$$\xi_{t|T}(i,j) = \frac{\gamma_{t|t}(i)a_{ij}}{\sum_{i=1}^{N} \gamma_{t|t}(i)a_{ij}} \gamma_{t+1|T}(j) \tag{2.15}$$

Although fixed-interval smoothing is the standard for batch learning, waiting until the last observation before providing the state estimates incurs long delays as well as a large memory complexity $\mathcal{O}(NT)$, to store the filtered state estimate for the entire observation sequence (see Khreich et al. (2010c) for more details). These issues are prohibitive for on-line learning.

A Forward Only (FO) fixed-interval smoothing algorithm has been proposed as an alternative to reduce the memory requirement of the FB or FFBS algorithms (Churbanov and Winters-Hilt, 2008; Elliott et al., 1995; Sivaprakasam and Shanmugan, 1995). The basic idea is to directly propagate all smoothed information in the forward pass:

$$\sigma_t(i,j,k) \triangleq \sum_{\tau=1}^{t-1} P(q_\tau = i, q_{\tau+1} = j, q_t = k \mid o_{1:t}, \lambda),$$

which represents the probability of having made a transition from state S_i to state S_j at some point in the past ($\tau < t$) and of ending up in state S_k at the current time t. At the next time step, $\sigma_{t+1}(i,j,k)$ can be recursively computed from $\sigma_t(i,j,k)$ using:

$$\sigma_t(i,j,k) = \sum_{n=1}^{N} \sigma_{t-1}(i,j,n) a_{nk} b_k(o_t) + \alpha_{t-1}(i) a_{ik} b_k(o_t) \delta_{jk} \tag{2.16}$$

from which the smoothed state densities can be obtained, at time t, by marginalizing over the states. The advantage of the FO algorithm is that it provides the exact smoothed state estimate at each time step with the linear memory complexity $\mathcal{O}(N)$. However, this is achieved at the expenses of increasing the time complexity to $\mathcal{O}(N^4 T)$ as can be seen in the four-dimensional recursion (2.16). As described in Section 2.3, several authors have proposed the FO algorithm for on-line learning of HMM parameters.

2.2.2 Optimization Techniques:

Maximum-likelihood (ML) parameter estimation in Hidden Markov Models (HMMs) can be carried out using either the expectation maximization (EM) (Baum et al., 1970; Dempster et al., 1977) or standard numerical optimization techniques (Nocedal and Wright, 2006; Zucchini and MacDonald, 2009). This section describes both estimation procedures for HMMs.

2.2.2.1 Expectation-Maximization:

The EM is a general iterative method to find the MLE of the parameters of an underlying distribution given a data set when the data is incomplete or has missing values. EM alternates between computing an expectation of the likelihood (E-step) – by including the latent variables as if they were observed – and a maximization of the expected likelihood (M-step) found on the E-step. The parameters found in the M-step are then used to initiate another iteration until a *monotonic* convergence to a stationary point of the likelihood (Dempster et al., 1977).

In the context of HMM, the basic idea consists of optimizing an auxiliary $\mathcal{Q}$-function (also known as the intermediate quantity of EM):

$$\begin{aligned}\mathcal{Q}_T(\lambda, \lambda^{(k)}) &= E_{\lambda^{(k)}}\left\{\log P(o_{1:T}, q_{1:T} \mid \lambda) \mid o_{1:T}, \lambda^{(k)}\right\} \\ &= \sum_{q \in S} P(o_{1:T}, q_{1:T} \mid \lambda^{(k)}) \log P(o_{1:T}, q_{1:T} \mid \lambda)\end{aligned} \tag{2.17}$$

which is the expected value of the *complete-data* log-likelihood. By assuming the probability values of the hidden states, the $\mathcal{Q}$-function is therefore easier to optimize than the *incomplete-data* log-likelihood (2.4). Nevertheless, it can be explicitly expressed in terms of HMM parameters. For instance, for a discrete output HMM it has the following form:

$$\mathcal{Q}_T(\lambda, \lambda^{(k)}) = \sum_{i=1}^{N} \gamma_{1|T}^{(k)}(i) \log \pi_i + \sum_{t=1}^{T-1} \sum_{i=1}^{N} \sum_{j=1}^{N} \xi_{t|T}^{(k)}(i,j) \log a_{ij} + \sum_{t=1}^{T} \sum_{j=1}^{M} \gamma_{t|T}^{(k)}(j) \delta_{o_t v_m} \log b_j(m) \tag{2.18}$$

Each term on the right hand side can be maximized individually using Lagrange multipliers subject to the relevant constraints (2.2). The solutions provide the same update formulas as those provided with the Baum-Welch algorithm below, (2.19), (2.20) and (2.21).

Starting with an initial guess $\lambda^{(0)}$, subsequent iterations, $k = 1, 2, \ldots$, of the EM consist in:

E-step: computing the auxiliary function $\mathcal{Q}(\lambda, \lambda^{(k)})$

M-step: determining the model that maximizes $\mathcal{Q}$, $\lambda^{(k+1)} = \arg\max_{\lambda} \mathcal{Q}(\lambda, \lambda^{(k)})$

If instead of maximizing $\mathcal{Q}$, we find some $\lambda^{(k+1)}$ that increases the likelihood (partial M-step) then it is called Generalized EM (GEM), which is also guaranteed to converge (Wu, 1983) .

The Baum-Welch (BW) algorithm (Baum and Petrie, 1966; Baum et al., 1970) is a specialized EM algorithm for estimating HMM parameters. Originally introduced to estimate the parameters when provided with a single discrete observations sequence (Baum et al., 1970), it was later extended for multiple observation sequences (Levinson et al., 1983; Li et al., 2000) and for continuous observations (Juang et al., 1986; Liporace, 1982). Given a sequence $o_{1:T}$ of T observations, an HMM with N states, and an initial guess $\lambda^{(0)}$, the BW algorithm proceeds as follows. During each iteration k, the FB or FF-BS computes the expected number of state transitions and state emissions based on the current model (E-step), and then re-estimated the model parameters (M-step) using:

$$\pi_i^{(k+1)} = \gamma_{1|T}^{(k)}(i) = \text{ (expected frequency of state } S_i \text{ at } t = 1) \tag{2.19}$$

$$a_{ij}^{(k+1)} = \frac{\sum_{t=1}^{T-1} \xi_{t|T}^{(k)}(i,j)}{\sum_{t=1}^{T-1} \gamma_{t|T}^{(k)}(i)} = \frac{\text{expected \#of trans. from } S_i \to S_j}{\text{expected \#of trans. from } S_i} \tag{2.20}$$

$$b_j^{(k+1)}(m) = \frac{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j) \delta_{o_t v_m}}{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j)} = \frac{\text{expected \#of times in } S_j \text{ observing } v_m}{\text{expected \#of times in } S_j} \tag{2.21}$$

Each iteration of the E-step and M-step is guaranteed to increase the likelihood of the observations giving the new model until a convergence to a stationary point of the likelihood (Baum, 1972; Baum et al., 1970; Dempster et al., 1977).

For a continuous HMM only the last term of (2.18) and hence (2.21) must be changed according to the state parametric density. For instance, for an HMM with Gaussian state

densities (2.1), the state outputs update are given by:

$$\hat{\mu_j} = \frac{\sum_{t=1}^{T} \gamma_t(j) o_t}{\sum_{t=1}^{T} \gamma_t(j)}, \text{ and } \hat{\sigma}_j^2 = \frac{\sum_{t=1}^{T} \gamma_t(j)(o_t - \mu_j)^2}{\sum_{t=1}^{T} \gamma_t(j)}$$

For multiple independent observation sequences, Levinson et al. (1983) proposed averaging the smoothed conditional densities, (2.14) and (2.14), over all observation sequences in the E-step, then updating the model parameters (M-step). Li et al. (2000) propose using a combinatorial method on individual observations probabilities, rather than their product, to overcome the independence assumption. For both cases, re-estimating parameters with the standard EM requires accessing all of the sequences in the training block during each iteration.

2.2.2.2 Standard Numerical Optimization:

While EM-based techniques indirectly optimize the log-likelihood function (2.4) through the complete-data log-likelihood (2.17), standard numerical optimization methods work directly with the log-likelihood function and its derivatives.

For instance, starting with an initial guess of the model λ^0, first order methods such as the gradient descent update the model at each iteration k using:

$$\lambda^{(k+1)} = \lambda^{(k)} + \eta_k \nabla_\lambda \ell_T(\lambda^{(k)}) \tag{2.22}$$

where the learning rate η_k decrease monotonically over training iterations to ensure that the sequence $\ell_T(\lambda^{(k)})$ is non-decreasing. It is usually chosen as to maximize the objective function in the search direction:

$$\eta_k = \arg\max_{\eta \geq 0} \ell_T[\lambda^{(k)} + \eta \nabla_\lambda(\lambda^{(k)})] \tag{2.23}$$

However due to its linear convergence rate, gradient descent methods could converge slowly in large optimization spaces.

Second order methods guarantee a faster convergence. For instance, the Newton-Raphson method updates the model at each iteration k using:

$$\lambda^{(k+1)} = \lambda^{(k)} - H^{-1}(\lambda^{(k)})\nabla_\lambda \ell_T(\lambda^{(k)}) \tag{2.24}$$

The convergence rate is at least quadratic at the convergence point, for which the Hessian is negative definite. However, if the initial parameters are far from those of true model parameters, the convergence is not guaranteed. A learning rate η_k similar to (2.23) can be introduced to control the step length in the search direction. A polynomial interpolation of $\ell_T(\lambda)$ along the line-segment between $\lambda^{(k)}$ and $\lambda^{(k+1)}$ is usually used in practice, since it is often impossible to have an exact maximum of the line search. Nonetheless, the Hessian $H = \nabla^2_\lambda \ell_T(\lambda^{(k)})$ could be non-invertible or not negative semi-definite.

To avoid these issues, quasi-Newton methods use an adaptive matrix $W^{(k)}$ which provides an approximation of the Hessian at each iteration:

$$\lambda^{(k+1)} = \lambda^{(k)} + \eta_k W^{(k)} \nabla_\lambda \ell_T(\lambda^{(k)}) \tag{2.25}$$

where $W^{(k)}$ is negative definite to ensure that ascent direction is chosen. The numerical issues associated with the matrix inversion are therefore avoided, while still exhibiting the convergence rate of the Newton algorithms near the convergence point.

For a discrete HMM, the gradient of the log-likelihood $\nabla_\lambda \ell_T(\lambda^{(k)})$ can be computed using the state conditional densities obtained from the fixed-interval smoothing algorithms (2.14) and (2.15) as:

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi^{(k)}_{t|T}(i,j)}{a_{ij}} \tag{2.26}$$

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial b_j(m)} = \frac{\sum_{t=1}^{T} \gamma^{(k)}_{t|T}(j)\delta_{o_t v_m}}{b_j(m)} \tag{2.27}$$

An alternative computation can be achieved by using a recursive computation of the gradient itself as described in sub-section 2.3.2.2. Once the gradient of the likelihood is computed, the HMM parameters are then additively updated according to, for instance, the first order (2.22) or second order (2.25) methods. For multiple independent observation sequences, the derivatives with reference to each model parameter (2.26) and (2.27) must be accumulated and averaged over all observation sequences, prior to updating the model parameters. The reader is referred to Cappe et al. (1998); Qin et al. (2000) for more details on quasi-Newton, and to Turner (2008) on Levenberg–Marquardt optimizations for HMMs.

Standard numerical optimization methods have to ensure parameter constraints, (2.2) and (2.3), explicitly through either a constrained optimization or a re-parametrization to reduce the problem to unconstrained optimization. Levinson et al. (1983) detail the early implementation of a constrained optimization for HMM using Lagrange multipliers. Among the unconstrained transformation is the soft-max parametrization proposed in (Baldi and Chauvin, 1994), which maps the bounded space (a, b) to the unbounded space (u, v):

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_k e^{u_{ik}}} \text{ and } b(k) = \frac{e^{v_j(k)}}{\sum_z e^{v_j(z)}} \tag{2.28}$$

Other unconstrained parametrization consist of a projection of the model parameters onto the constraints domain, such as a simplex or a sphere. The parametrization on a simplex (Krishnamurthy and Moore, 1993; Slingsby, 1992)

$$\triangle = \{a_{ij} \in \mathbb{R}^{N-1} : \sum_{j=1}^{N-1} a_{ij} \leq 1, \text{ and } a_{ij} \geq 0\} \tag{2.29}$$

enforces the stochastic constraints (2.2) by updating all but one of the parameters in each row, $a_{l_i} = 1 - \sum_{j \neq l_i}^{N} a_{ij}$, $1 \leq l_i \leq N$. However, it does not ensure the positiveness of the parameters (2.3). One improvement consists in using a restricted projection to the

space of positive matrices for some positive ϵ (Arapostathis and Marcus, 1990):

$$\triangle_\epsilon = \{a_{ij} \in \mathbb{R}^{N-1} : \sum_{j=1}^{N-1} a_{ij} \leq 1-\epsilon, \text{ and } a_{ij} \geq \epsilon\} \tag{2.30}$$

Alternatively, parametrization on a sphere adequately enforces both constraints (Collings et al., 1994):

$$\mathbb{S}^{N-1} = \{s_{ij} : \sum_{j=1}^{N} s_{ij}^2 = 1\} \tag{2.31}$$

where $a_{ij} = s_{ij}^2$. This also has the advantage that the constraint manifold is differentiable at all points.

2.2.2.3 Expectation-Maximization versus Gradient-based Techniques

For discrete output HMMs, the EM algorithm is generally easier to apply than gradient-based technique since derivatives, Hessian inversion, line-search, etc., are not required. It is numerically more robust against poor initialization values or when the model has large number of parameters. It also has a monotonic convergence propriety which is not maintained in gradient-based techniques, and ensures parameters constraints implicitly. However the rate of convergence of the EM can be very slow, it is usually linear in the neighborhood of a maximum (Dempster et al., 1977). This is comparable to gradient descent but slower than the quadratic convergence that can be achieved with second order or conjugate gradient techniques. Nevertheless, EM steps do not always involve closed-form expressions in such cases the maximization has to be carried out numerically. Another advantage of numerical optimization techniques is that they automatically yield an estimate of parameters variances (Cappe and Moulines, 2005; Zucchini and MacDonald, 2009). Since there is no clear advantage of one technique on the other, hybrid algorithms have been proposed to take advantage from both techniques (Bulla and Berzel, 2008; Lange, 1995).

Given a block of data with R independent sequences each of length T and an N state HMM, the time an complexity per iteration for EM and gradient-based techniques are

$\mathcal{O}(RN^2T)$ and their memory complexity is $\mathcal{O}(NT)$. This is because they both rely on the fixed-interval smoothing algorithms (2.14) and (2.15) to compute the state densities for the E-step or the gradient of the log-likelihood. The difference is basically related to the internal procedures employed in gradient-based such as the line search, and to the speed of convergence (see Cappe and Moulines (2005, Ch. 10) for more details).

2.3 On-Line Learning of HMM Parameters

Several on-line learning techniques from the literature may be applied to incremental learning of HMM parameters from new training sequences. Figure 2.4 presents a taxonomy of techniques for on-line learning of HMM parameters, according to objective function, optimization technique, and target application. As shown in the figure, they fall in the categories of standard numerical optimization, expectation-maximization and recursive estimation, with the objective of either maximizing the likelihood estimation (MLE) criterion, minimizing the model divergence (MMD) of parameters penalized with the MLE, or minimizing the output or state prediction error (MPE).

The target application implies a scenario for data organization and learning. Some techniques have been designed for *block-wise* estimation of HMM parameters, while others for *symbol-wise* estimation of parameters. Block-wise techniques are designed for scenarios in which training symbols are organized into a block of sub-sequences and the HMM re-estimates its parameters after observing each sub-sequence. In contrast, symbol-wise techniques, also known as as recursive or sequential techniques, are designed for scenarios in which training symbols are observed one at a time, from a stream of symbols, and the HMM parameters are re-estimated upon observing each new symbol. The rest of this section provides a survey of techniques for on-line learning of HMM parameters shown in Figure 2.4.

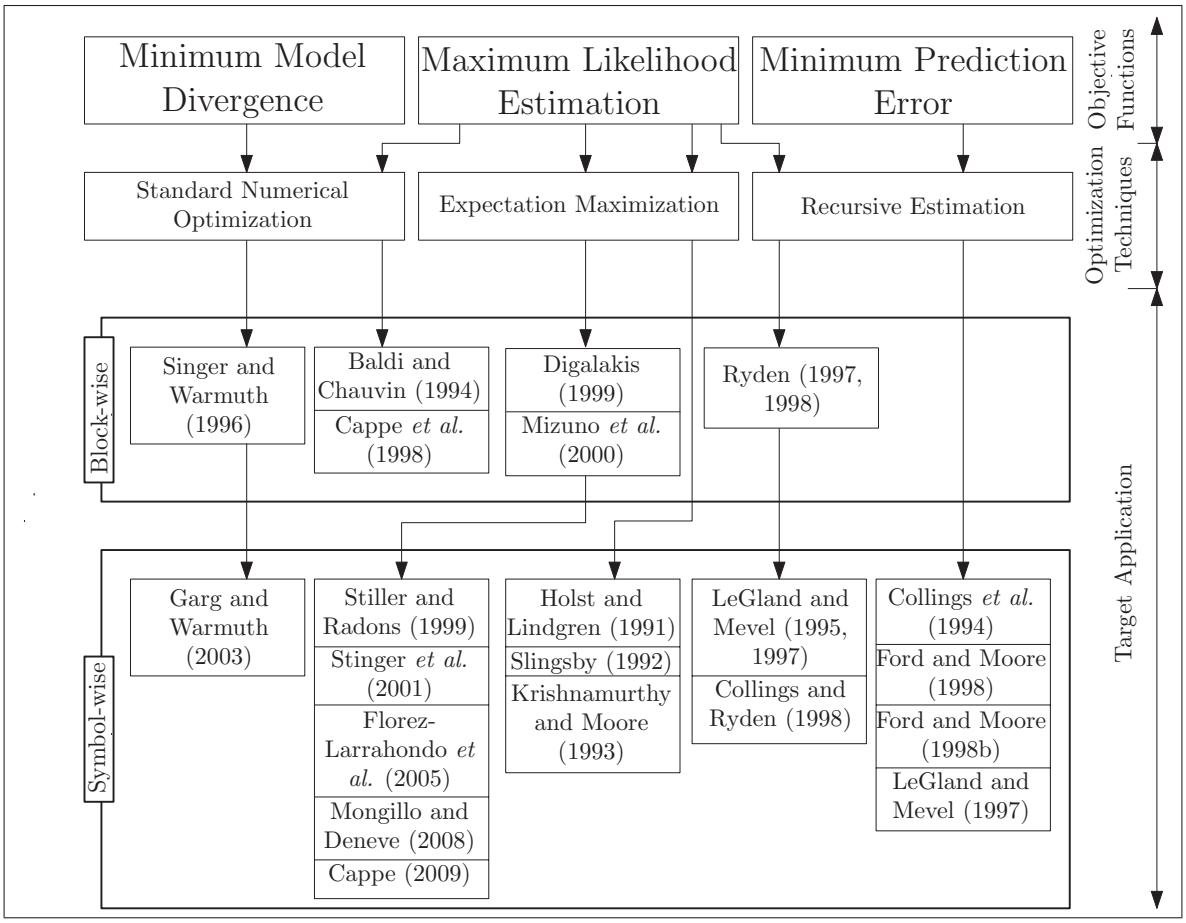


Figure 2.4 Taxonomy of techniques for on-line learning of HMM parameters

2.3.1 Minimum Model Divergence (MMD)

A dual cost function that maximizes the log-likelihood while minimizing the divergence of model parameters, using an exponentiated gradient optimization framework (Kivinen and Warmuth, 1997), is first proposed for the block-wise case (Singer and Warmuth, 1996), then extended to symbol-wise case (Garg and Warmuth, 2003).

Block-wise

Based on the exponentiated gradient framework (Kivinen and Warmuth, 1997), Singer and Warmuth (1996) proposed an objective function for discrete HMMs that minimizes the divergence between old and new HMM parameters penalized by the negative log-

likelihood of each sequence multiplied by a fixed positive learning rate ($\eta > 0$):

$$\lambda^{(k+1)} = \arg\min_{\lambda} \Big(KL(\lambda^{(k+1)}, \lambda^{(k)}) - \eta \ell_T(\lambda^{(k+1)}) \Big)$$

where $KL(\lambda^{(k+1)}, \lambda^{(k)})$ is the Kullback-Leibler (KL) divergence between the distributions $\lambda^{(k+1)}$ and $\lambda^{(k)}$. KL is defined between two probability distributions $P_{\lambda^0}(o_{1:t})$ and $P_{\lambda}(o_{1:t})$ by

$$KL_t(P_{\lambda^0} \parallel P_{\lambda}) = E\left\{ \log \frac{P_{\lambda^0}(o_{1:t})}{P_{\lambda}(o_{1:t})} \right\} = \sum_t P_{\lambda^0}(o_{1:t}) \log \frac{P_{\lambda^0}(o_{1:t})}{P_{\lambda}(o_{1:t})} \tag{2.32}$$

which is always non-negative and attains its global minimum at zero for $P_{\lambda} \to P_{\lambda^{(0)}}$. Its limit ($t \to \infty$) is Kullback-Leibler rate or relative entropy rate.

The model parameters are updated after processing each observation sequence:

$$a_{ij}^{(k+1)} = \frac{1}{Z_1} a_{ij}^{(k)} \exp\left(-\frac{\eta}{n_i(\lambda^{(k+1)})} \frac{\partial \ell_T(\lambda^{(k+1)})}{\partial a_{ij}} \right) \tag{2.33}$$

$$b_j^{(k+1)}(m) = \frac{1}{Z_2} b_j^{(k)}(m) \exp\left(-\frac{\eta}{n_j(\lambda^{(k+1)})} \frac{\partial \ell_T(\lambda^{(k+1)})}{\partial b_j(m)} \right) \tag{2.34}$$

where Z_1 and Z_2 are normalization factors. The expected usage of state i, $n_i(\lambda^{(k)}) = \sum_{t=1}^{T} \gamma_{t|T}(i)$ can be computed using the filtering estimation (2.14), and the derivatives of the log-likelihood are given by (2.26) and (2.27).

Symbol-wise

Garg and Warmuth (2003) extended the block-wise learning of Singer and Warmuth (1996) to symbol-wise. The model divergence is however penalized by the negative of the log-likelihood increment, i.e., log-likelihood of each new symbol given all previous ones. At each time instant the model is updated using:

$$\lambda_{t+1} = \arg\min_{\lambda} \Big(KL(\lambda_{t+1}, \lambda_t) - \eta \log P(o_{t+1} \mid o_{1:t}, \lambda_{t+1}) \Big)$$

which can be decoupled into similar update, for both state transition and state output, as with (2.33) and (2.34), respectively. The main difference resides however in the computation of the expected usage of state i – it is computed recursively based on each symbol with $n_i(\lambda_t) = \sum_{\tau=1}^{t} \gamma_{\tau|t}(i)$, is the gradient of the log-likelihood increment. The recursive formulas proposed to implement this computation are very similar to the recursive maximum likelihood estimation proposed by LeGland and Mevel (LeGland and Mevel, 1995, 1997) (see Section 2.3.2).

2.3.2 Maximum Likelihood Estimation (MLE)

The attractive asymptotic properties of the MLE in general, have prompted many attempts to extend this objective function to on-line learning for different applications. In particular, indirect maximization of log-likelihood, through the complete log-likelihood (2.17), has been the basis for many on-line learning algorithms. Neal and Hinton (1998) have proposed an "incremental" version of the EM algorithm to accelerate its convergence on a finite training data set[4]. Assuming a fixed training data set is divided into n blocks, each iteration of this algorithm performs a partial E-step (for a selected block) then updates the model parameters (M-step), until a convergence criterion is met. The improved convergence is attributed to exploiting new information more quickly, rather than waiting for the complete data to be processed before updating the parameters (Gotoh et al., 1998; Neal and Hinton, 1998). Many extensions to this algorithm have emerged for on-line block-wise (Digalakis, 1999; Mizuno et al., 2000) or symbol-wise (Florez-Larrahondo et al., 2005; Stenger et al., 2001; Stiller and Radons, 1999) learning.

Direct maximization of the log-likelihood function is first proposed for the block-wise case using the GD algorithm (Baldi and Chauvin, 1994), and then using the quasi-Newton algorithm (Cappe et al., 1998) for faster convergence. A recursive block-wise estimation technique is also proposed for this optimization (Ryden, 1998, 1997). Finally, extensions to the work of Titterington (1984) and Weinstein E. and Oppenheim (1990)

[4]In contrast, the incremental scenario considered in this chapter restricts accessing a block of data once it has been processed.

for independent data has lead to a symbol-wise technique to optimize the complete data likelihood (2.17) recursively, at each time step.

2.3.2.1 On-line Expectation-Maximization

An important property of the "incremental" version of EM resides in its flexibility (Neal and Hinton, 1998). There is no constraints on how the data is divided into blocks. The block might range from a single observation symbol to one or multiple sequence of observations. In addition, one can choose to update the parameters by selecting data blocks cyclically, or by any other scheme. Accordingly, several algorithms have extended this approach to on-line learning of HMM parameters. As detailed next, this is basically achieved by initializing the smoothed densities to zero, processing the training data (symbol-wise or block-wise) sequentially, and updating the HMM after each symbol or sequence.

Block-wise

Digalakis (1999) derived an on-line algorithm of EM to update the parameters of a continuous HMM for automatic speech recognition, and showed faster convergence and higher recognition rate over the batch algorithm. Similarly, Mizuno et al. (2000) proposed a similar algorithm for a discrete HMM however. Given the initial model λ_0, and after processing each new sequence of observation of length T, the state densities are recursively computed using:

$$\sum_{t=1}^{T-1} \xi_{t|T}^{(k+1)}(i,j) = (1-\eta_k) \sum_{t=1}^{T-1} \xi_{t|T}^{(k)}(i,j) + \eta_k \sum_{t=1}^{T-1} \xi_{t|T}^{(k+1)}(i,j)$$

$$\sum_{t=1}^{T} \gamma_{t|T}^{(k+1)}(j)\delta_{o_t m} = (1-\eta_k) \sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j)\delta_{o_t m} + \eta_k \sum_{t=1}^{T} \gamma_{t|T}^{(k+1)}(j)\delta_{o_t m}$$

and the model parameters are then directly updated using (2.20) and (2.21). The learning rate η_k is proposed in polynomial form $\eta_k = c(\frac{1}{k})^d$ for some positive constants c and d.

Symbol-wise

Stiller and Radons (1999) introduced a recursive algorithm for non-stationary discrete Mealy HMMs[5], which is essentially based on the FO (2.16). The main idea is to recursively update a tensorial quantity,

$$\xi^t_{ijk}(o_t) = \sum_{\tau=1}^{t} \alpha_i^{\tau-1} a_{ij}^{\tau-1}(o_\tau) \beta_{jk}^{\tau,t} \delta_{o_t o_\tau} \tag{2.35}$$

which accounts for the weighted sum of transition probability from state S_i to S_j (at time τ), emitting symbol o_t and being in state k at time $t \geq \tau$. The model parameters (emission on arcs) are condensed in $a^t_{ij}(o_t) = P(q_t = j, o_t \mid q_{t-1} = i)$. The forward-vector ($\alpha^\tau$) and backward-matrix ($\beta^{\tau,t}$) are recursively computed by:

$$\alpha^\tau = \pi \prod_{r=1}^{\tau} a^{(r-1)}(o_r) \text{ and } \beta^{\tau,t} = \prod_{r=\tau+1}^{t} a^{r-1}(o_r)$$

Equation (2.35) can therefore be recursively computed as function of the old $\xi^t_{ijk}(o_t)$ and the new symbol o_{t+1}:

$$\xi^{t+1}_{ijk}(o_{t+1}) = \sum_{\hat{k}} \xi^t_{ij\hat{k}}(o_t) \frac{\sum_l \xi^t_{\hat{k}kl}(o_{t+1})}{\sum_\tau \sum_{\hat{\jmath},\hat{l}} \xi^t_{i\hat{\jmath}\hat{l}}(o_\tau)} + \eta_t \delta_{o_t o_{t+1}} \delta_{jk} \sum_{l,m} \sum_\tau \xi^t_{lmi}(o_\tau) \frac{\sum_k \xi^t_{ijk}(o_t)}{\sum_\tau \sum_{\hat{\jmath},\hat{k}} \xi^t_{ij\hat{k}}(o_\tau)}$$

where η_t is a time-varying learning rate. Finally, the model parameters (a^τ_{ij}) and the probability of the system currently being in state k (α^τ_k) can be estimated at any time using:

$$a^t_{ij}(o) = \frac{\sum_k \xi^t_{ijk}(o_t)}{\sum_t \sum_{\hat{\jmath},\hat{k}} \xi^t_{ij\hat{k}}(o_t)} \text{ and } \alpha^\tau_k = \frac{\sum_{i,j} \sum_t \xi^t_{ijk}(o_t)}{\sum_{i,j,\hat{k}} \sum_t \xi^t_{ij\hat{k}}(o_t)}$$

This algorithm have been recently proposed for Moore HMMs (Mongillo and Deneve, 2008) and then generalized (Cappe, 2009).

[5]The emission is produced on transitions (Mealy model) while for all other algorithms presented the output is produced on states (Moore model).

Stenger et al. (2001) approximated the fixed-interval smoothing, used in BW, by the filtered state density (2.9) for updating the parameters of a continuous HMM from a stream of data. Naturally, the filtered state density is recursively computed independently of the sequence length. The model parameters are then updated, at each time step, by

$$\begin{aligned} a_{ij}^{t} &= \frac{\sum_{\tau=1}^{t-2}\gamma_{t|\tau}(i)}{\sum_{\tau=1}^{t-1}\gamma_{t|\tau}(i)}a_{ij}^{t-1} + \frac{\xi_{t-1|t}(i,j)}{\sum_{\tau=1}^{t-1}\gamma_{t|\tau}(i)} \\ \mu_{i}^{t} &= \frac{\sum_{\tau=1}^{t-1}\gamma_{t|\tau}(i)}{\sum_{\tau=1}^{t}\gamma_{t|\tau}(i)}\mu_{i}^{t-1} + \frac{\gamma_{t|t}(i)o_t}{\sum_{\tau=1}^{t}\gamma_{t|\tau}(i)} \\ \left[\sigma_{i}^{2}\right]^{t} &= \frac{\sum_{\tau=1}^{t-1}\gamma_{t|\tau}(i)}{\sum_{\tau=1}^{t}\gamma_{t|\tau}(i)}\left[\sigma_{i}^{2}\right]^{t-1} + \frac{\gamma_{t|t}(i)(o_t-\mu_{i}^{t})^{2}}{\sum_{\tau=1}^{t}\gamma_{t|\tau}(i)} \end{aligned} \tag{2.36}$$

An exponential forgetting is proposed to make old estimates receive less weight by tuning a fixed learning rate.

Similarly, Florez-Larrahondo et al. (2005) proposed, for discrete HMM, using the predictive state density (2.10) as a better approximation[6]. The model parameters are recursively updated using an adaptation of Stenger's recursion formulas for discrete HMMs however. The transition update is the same as (2.36) with $\gamma_{t|\tau}$ replaced by $\gamma_{t+1|\tau}$, while the state output is given by:

$$b_{j}^{T}(k) = \frac{\sum_{\tau=1}^{t-1}\gamma_{t+1|\tau}(j)}{\sum_{\tau=1}^{t}\gamma_{t+1|\tau}(j)}b_{j}^{T-1}(k) + \frac{\gamma_{t+1|t}(j)\delta(o_T,v_k)}{\sum_{\tau=1}^{t}\gamma_{t+1|\tau}(j)} \tag{2.37}$$

The author argued that there is no need to exponential forgetting and proposed postponing the first update of the parameters until some time $t>0$ so that enough statistics should have been collected to stabilize the re-estimation.

[6]This is equivalent to setting $\beta_t(i)=\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})$ in the FB algorithm.

2.3.2.2 Numerical Optimization Methods:

Block-wise

Based on the GD of the negative likelihood (2.22), Baldi and Chauvin (1994) have introduced a block-wise algorithm for estimation of discrete HMM parameters, which related to the Generalized EM (GEM). By using the soft-max parametrization, (2.28), and the FF-BS algorithm, parameters are updated after each sequence as follows:

$$\begin{aligned} u_{ij}^{(k+1)} &= u_{ij}^{(k)} + \eta \sum_{t=1}^{T} \Big(\xi_{t|T}(i,j) - a_{ij} \gamma_{t|T}(i) \Big) \\ v_j^{(k+1)}(m) &= v_j^{(k+1)}(m) + \eta \sum_{t=1}^{T} \Big(\gamma_{t|T}(j) \delta_{o_t m} - b_j(m) \gamma_{t|T}(j) \Big) \end{aligned}$$

This is typically referred to as stochastic gradient-based techniques (Bottou, 2004).

Based on the recursive maximum likelihood estimation of LeGland and Mevel (LeGland and Mevel, 1995, 1997) presented below, Cappe et al. (1998) proposed a quasi-Newton (2.25) faster convergent technique for maximizing the likelihood of a continuous HMM with Gaussian output (2.1). Using the parametrization on a simplex (2.29) and replacing the standard deviation by its log (to ensure its positivity), the gradient of the log-likelihood (2.13), can be computed recursively using:

$$\nabla_\lambda \ell_T(\lambda^{(k)}) = \sum_{t=1}^{T} \frac{1}{c_t} \sum_{i=1}^{N} \Big[\gamma_{t|t-1}(i) \nabla_{\lambda^{(k-1)}} b_i(o_t) + b_i(o_t) \nabla_{\lambda^{(k-1)}} \gamma_{t|t-1}(i) \Big] \tag{2.38}$$

where $c_t = \sum_{k=1}^{N} b_k(o_t) \gamma_{t|t-1}(k)$ is a normalization factor. The gradient of the predictive density is also computed recursively, with reference to the model parameters, using:

$$\frac{\partial \gamma_{t+1|t}(j)}{\partial a_{ij}} = \frac{1}{c_t} \sum_{i=1}^{N} \left[\Big(a_{ij} - \gamma_{t+1|t}(j) \Big) \frac{\partial \gamma_{t|t-1}(i)}{\partial a_{ij}} + \gamma_{t|t-1}(i) \right]$$

$$\nabla_{\lambda_b} \gamma_{t+1|t}(j) = \frac{1}{c_t} \sum_{i=1}^{N} \Big(a_{ij} - \gamma_{t+1|t}(j) \Big) \Big[\gamma_{t|t-1}(i) \nabla_{\lambda_b} b_i(o_t) + b_i(o_t) \nabla_{\lambda_b} \gamma_{t|t-1}(i) \Big]$$

and the gradient of the Gaussian output density (2.1), w.r.t. the mean and the standard deviation, is given by

$$\nabla_{\lambda_b} b_i(o_t) = \left[\frac{\partial b_i(o_t)}{\partial \mu_i}, \frac{\partial b_i(o_t)}{\partial \sigma_i}\right]' = \left[\frac{(o_t - \mu_i)}{\sigma_i^2} b_i(o_t), \frac{1}{\sigma_i}\left(\left(\frac{(o_t - \mu_i)}{\sigma_i}\right)^2 - 1\right) b_i(o_t)\right]' \quad (2.39)$$

2.3.2.3 Recursive Maximum Likelihood Estimation (RMLE):

The general recursive estimator for the parameters of a stochastic process is of the form (Benveniste et al., 1990; Ljung and Soderstrom, 1983):

$$\lambda_{t+1} = \lambda_t + \eta_t H(o_{t+1}, \lambda_t)^{-1} h(o_{t+1}, \lambda_t) \quad (2.40)$$

where η_t is a sequence of small gains (constant or decreasing with t), $H(o_{t+1}, \lambda_t)^{-1}$ an adaptive matrix, and $h(o_{t+1}, \lambda_t)$ is a score function. Both the adaptive matrix, H, and the score function, h, determine the update of the parameter λ_t as a function of new observations. The goal of the recursive procedures is to find the roots of a regression function given the true parameters λ^0: $\lim_{t\to\infty} E\left\{H(o_{t+1}, \lambda_t)^{-1} h(o_{t+1}, \lambda_t) \mid \lambda^0\right\}$. In the general case of incomplete data, the score is equal to the gradient of the log-likelihood function $h = \nabla_\lambda \log P(o_{t+1} \mid \lambda)$. The adaptive matrix is equal to the incomplete data observed information matrix $H = \nabla^2_\lambda \log P(o_{1:t+1} \mid \lambda)$, which ideally must be taken as the incomplete data Fisher information matrix $H = E_\lambda\left[h(o_{t+1} \mid \lambda) h'(o_{t+1} \mid \lambda)\right]$. However, for the incomplete data models, explicit form of the incomplete data Fisher information matrix is rarely available. Titterington (1984, Eq. 9) proposed to use the complete data Fisher information matrix instead $H = E\left[-\nabla^2_{\lambda_t} \log P(o_t, q_t \mid \lambda)\right]$. He showed that it is easy to compute and invert because it is always block-diagonal with respect to latent data and model parameters. For several independent and identically distributed (i.i.d.) incomplete data models, this recursion had proven to be consistent and asymptotically

normal (Titterington, 1984):

$$\lambda_{t+1} = \lambda_t + \frac{1}{t+1}\left(E\left[-\nabla^2_{\lambda_t} \log P(o_t, q_t \mid \lambda)\right]\right)^{-1} \nabla_{\lambda_t} \log P(o_{t+1} \mid \lambda_t) \tag{2.41}$$

This recursive learning algorithm is related to on-line estimation of parameters using EM. It was also related to an EM iteration since it uses a recursive version of (2.17).

The efficiency and convergence is an issue with Titterington's recursion (Ryden, 1998; Wang and Zhao, 2006). In fact, for HMMs, the score function must consider the previous observations:

$$h(o_{t+1} \mid \lambda) = \nabla_\lambda \log P(o_{t+1} \mid o_{1:t}, \lambda) \tag{2.42}$$

Another recursion using the relative entropy (2.32) as objective function, is proposed by Weinstein E. and Oppenheim (1990, Eq. 4):

$$\hat{\lambda}_{t+1} = \hat{\lambda}_t + \eta_t h(o_{t+1} \mid \hat{\lambda}_t) \tag{2.43}$$

where the gain η_t satisfies:

$$\lim_{t\to\infty} \eta_t = 0;\ \sum_{t=1}^{\infty} \eta_t = \infty;\ \sum_{t=1}^{\infty} \eta_t^2 \leq \infty \tag{2.44}$$

It was suggested that h may be calculated from the complete data using

$$h(o_{t+1} \mid \hat{\lambda}_t) = E_\lambda \left\{\nabla \log P(o_{t+1}, q_{t+1} \mid \lambda) \mid o_{t+1}\right\}$$

and is shown to be consistent in the strong sense and in the mean-square sense for stationary ergodic processes that satisfy some regularity conditions. However, some of these conditions do not hold for HMM (Ryden, 1998, 1997).

Block-wise

The idea proposed by Ryden (Ryden, 1998, 1997) is to consider successive blocks of m observations taken from data stream, $\boldsymbol{o}_n = \{o_{(n-1)m+1}, \ldots, o_{nm}\}$, as independent of each other, while the Markov dependence is only maintained within the observations of each block. This assumption reduces the extraction of information from all the previous observations to a data-segment of length m. Although it gives an approximation of the MLE – a less efficient maximum pseudo-likelihood – the presented technique is easier to apply in practice. Using a projection $\mathbb{P}_G$ on the simplex (2.30), $\triangle_\epsilon$, for some $\epsilon > 0$, at each iteration, the recursion is given (without matrix inversion)

$$\hat{\lambda}_{n+1} = \mathbb{P}_G \left(\hat{\lambda}_n + \eta_n h(\boldsymbol{o}_{n+1} \mid \hat{\lambda}_n) \right) \tag{2.45}$$

where $h(\boldsymbol{o}_{n+1} \mid \hat{\lambda}_n) = \nabla_{\lambda_n} \log P(\boldsymbol{o}_{n+1} \mid \lambda_n)$ and $\eta_n = \eta_0 n^{-\rho}$ for some positive constant η_0 and $\rho \in (\frac{1}{2}, 1]$. It was shown to converge almost surely to the set of Kuhn-Tucker points for minimizing the relative entropy $KL_m(\lambda^0 \parallel \lambda)$ defined in (2.32), which attains its global minimum at $\hat{\lambda}_n \to \lambda^0$ provided that the HMM is identifiable (Leroux, 1992; Ryden, 1994), hence the requirement $m \geq 2$.

Symbol-wise

Holst and Lindgren (1991, Eq. 16) proposed a similar recursion to (2.41) for estimating the parameter of an HMM however. They used the conditional expectation over (q_{t-1}, q_t) given $o_{1:t}$, which can be efficiently calculated using a forward recursion. It is guided by

$$H_t^{-1} = \frac{1}{t} \sum_{\tau=1}^{t} h(o_t \mid \hat{\lambda}_{t-1}) h'(o_t \mid \hat{\lambda}_{t-1})$$

which is different from the score function (2.42). The adaptive matrix is suggested by an empirical estimate to the incomplete data information matrix given by

$$H_t^{-1} = \frac{1}{t}\sum_{\tau=1}^{t} h(o_t \mid \hat{\lambda}_{t-1}) h'(o_t \mid \hat{\lambda}_{t-1})$$

which can be computed recursively without matrix inversion using:

$$H_t = \frac{t}{t-1}\left(H_{t-1} - \frac{H_{t-1} h_t h_t' H_{t-1}}{t-1+h_t' H_{t-1} h_t}\right)$$

Ryden (1997) argued that the recursion of Holst and Lindgren aims at local minimization of the relative entropy rate (2.32) between λ^0 and $\hat{\lambda}_t$. Moreover, he showed that if $\hat{\lambda}_{t+1} \to \lambda^0$, then $\sqrt{t}\left(\hat{\lambda}_{t+1} - \lambda^0\right)$ is asymptotically normal with zero mean and covariance matrix given by the inverse of this expectation $\lim_{t\to\infty} E\left\{h(o_{t+1}, \lambda^0) h'(o_{t+1}, \lambda^0)\right\}$.

Slingsby (1992) and Krishnamurthy and Moore (1993) derived an on-line algorithm for HMM based on the recursive version of the EM (2.41) proposed in (Titterington, 1984; Weinstein E. and Oppenheim, 1990). This technique applies a two step procedure, similar to EM, as each observation symbol arrives:

E-step: Recursively computes the complete data likelihood

$$\mathcal{Q}_{t+1}(\lambda, \lambda_t) = E_\lambda\left[\log P(o_{t+1}, q_{t+1} \mid \lambda) \mid o_{1:t+1}, \lambda_t\right] = \sum_{\tau=1}^{t+1} \mathcal{Q}_\tau(\lambda, \lambda_t) \tag{2.46}$$

M-step: Estimates $\lambda_{t+1} = \arg\max_\lambda \mathcal{Q}_{t+1}(\Lambda_t, \lambda)$

where $\mathcal{Q}_\tau(\lambda, \lambda_t)$ is defined in (2.17). The model is updated using:

$$\lambda_{t+1} = \lambda_t + \left[\nabla^2_{\lambda_t} \mathcal{Q}_{t+1}(\lambda, \lambda_t)\right]^{-1} \nabla_{\lambda_t} \mathcal{Q}_{t+1}(\lambda, \lambda_t)$$

Since each term of (2.46) depends on only one of the model parameters, $\nabla^2_{\lambda_t} \mathcal{Q}_{t+1}(\lambda, \lambda_t)$ is a block diagonal matrix for each of the model parameters. A projection into the constraint

domain along with the application of smoothing (fixed-lag or more conveniently sawtooth-lag) and forgetting show, through empirical experiments, a convergence to the correct model.

Slingsby (1992) presented the parameters update for a discrete HMM using a projection on a simplex (2.29):

$$a_{ij}^{(t+1)} = a_{ij}^{(t)} + \frac{1}{d_j^{(i)}} \left(g_j^{(i)} - \frac{\sum_{h=1}^{N} g_h^{(i)} / d_h^{(i)}}{\sum_{h=1}^{N} 1/d_h^{(i)}} \right) \tag{2.47}$$

where

$$d_j^{(i)} = \frac{\sum_{\tau=1}^{t+1} \xi_\tau(i,j)}{\hat{a}_{ij}^2}; \quad g_j^{(i)} = \frac{\xi_{t+1}(i,j)}{\hat{a}_{ij}} \tag{2.48}$$

$$b_j^{(t+1)}(k) = b_j^{(t)}(k) - b_j(o_{t+1}) \left(\frac{\gamma_{t+1}(j)}{\sum_{\tau=1}^{t+1} \gamma_\tau(j) \delta(o_{t+1}, k)} \right) \left(\frac{\frac{b_j(k)^2}{\sum_{\tau=1}^{t+1} \gamma_\tau(j) \delta(o_{t+1}, k)}}{\sum_{p=1}^{M} \left(\frac{b_j(p)^2}{\sum_{\tau=1}^{t+1} \gamma_\tau(j) \delta(o_{t+1}, p)} \right)} \right);$$

$$k \neq o_{t+1}$$

$$b_j^{(t+1)}(k) = 1 - \sum_{p \neq k}^{M} b_j^{(t+1)}(p);\ k = o_{t+1}$$

Krishnamurthy and Moore (1993) focused on continuous HMM with Gaussian output (2.1). The state transitions update is the same as (2.48) and (2.47), and the state output update is given by:

$$\mu_i^{(t+1)} = \mu_i^{(t)} + \frac{\gamma_{t+1}(i) \left(o_{t+1} - \mu_i^{(t)} \right)}{\sum_{\tau=1}^{t+1} \gamma_{\tau|t+1}(i)}$$

$$\left[\sigma_w^2\right]^{(t+1)} = \left[\sigma_w^2\right]^{(t)} + \frac{\sum_{i=1}^{N} \gamma_{t+1}(i) \left(o_{t+1} - \mu_i^{(t)} \right)^2 - \left[\sigma_w^2\right]^{(t)}}{t+1}$$

Since the parameters positiveness is not maintained with the simplex parametrization, the projection on a sphere (2.31) as detailed in (Collings et al., 1994) solves this issue.

Now

$$d_j^{(i)} = \sum_{\tau=1}^{t+1} \left[\frac{2\xi_\tau(i,j)}{s_{ij}^2} + 2\gamma_{t+1}(i) \right]; \quad g_j^{(i)} = \frac{2\xi_{t+1}(i,j)}{s_{ij}} - 2\gamma_{t+1}(i)s_{ij}$$

and the transition parameters update is given by

$$s_{ij}^{(t+1)} = s_{ij}^{(t)} + \frac{g_j^{(i)}}{d_j^{(i)}},\ a_{ij} = s_{ij}^2$$

LeGland and Mevel (1995, 1997) suggested and proved the convergence and the asymptotic normality of the RMLE (and the RCLSE described in the next section) without any stationary assumption using the geometric ergodicity and the exponential forgetting of their prediction filter and its gradient. This approach is based on the observation that the log likelihood can be expressed as an additive function of an extended Markov chain, i.e., as sum of terms depending on the observations and the state predictive filter (2.10):

$$\ell_\tau(\lambda) = \sum_{t=1}^{\tau} \log \sum_{i=1}^{N} b_i(o_t)\gamma_{t|t-1}(i)$$

Taking the gradient of the log-likelihood increment gives the score: (as a function of the extended Markov chain $Z_t = F(o_t, \gamma_{t|t-1}, w_t)$)

$$h(Z_t \mid \lambda) = \nabla_\lambda \ell_\tau(\lambda) = \frac{1}{c_t} \sum_{i=1}^{N} \left[\gamma_{t|t-1}(i)\nabla_\lambda b_i(o_t) + b_i(o_t) w_t^i \right]$$

where $c_t = \sum_{k=1}^{N} b_k(o_t)\gamma_{t|t-1}(k)$ is a normalization factor, and $w_t^i = \nabla_\lambda \gamma_{t|t-1}(i)$ is the gradient of the state prediction filter, which can be also computed recursively using:

$$w_{t+1}^i = R_1(o_t, \gamma_{t|t-1}, \lambda) w_t^i + R_2^i(o_t, \gamma_{t|t-1}, \lambda) \tag{2.49}$$

where

$$
\begin{aligned}
R_1(o_t, \gamma_{t|t-1}, \lambda) &= \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}\left[1 - \frac{\gamma_{t|t-1}(i)\sum_{k=1}^{N} b_k(o_t)}{c_t}\right]\frac{b_i(o_t)}{c_t} \\
R_2^i(o_t, \gamma_{t|t-1}, \lambda) &= \sum_{j=1}^{N} a_{ij}\left[1 - \frac{b_i(o_t)\sum_{k=1}^{N}\gamma_{t|t-1}(k)}{c_t}\right]\frac{\gamma_{t|t-1}(i)}{c_t}\nabla_\lambda b_i(o_t) + \\
&\quad \frac{\gamma_{t|t-1}(i)b_i(o_t)}{c_t}\sum_{j=1}^{N}\nabla_\lambda a_{ij}
\end{aligned}
$$

The parameters update is then done for each row i using:

$$
\hat{\lambda}_{t+1}^i = \mathbb{P}_\epsilon\left(\hat{\lambda}_t^i + \eta_{t+1} h(Z_t^i \mid \lambda)\right)
$$

where, $\eta_t = \frac{1}{t}$, and $\mathbb{P}_\epsilon$ is the projection on the simplex (2.30), $\triangle_\epsilon$, for some $\epsilon > 0$.

The update can be decoupled into transition and state output probabilities as follow (R_1 does not change). For transition probabilities, $\nabla_\lambda b_i(o_t)$ are zeros, accordingly h and R_2 reduces to:

$$
\begin{aligned}
h(Z_t \mid \lambda) &= \frac{1}{c_t}\sum_{i=1}^{N} b_i(o_t) w_t(i) \\
R_2^i(o_t, \gamma_{t|t-1}, \lambda) &= \frac{\gamma_{t|t-1}(i)b_i(o_t)}{c_t}\sum_{j=1}^{N}\nabla_\lambda a_{ij}
\end{aligned}
$$

where, $\nabla_\lambda a_{ij} = 1$ at position i, j and zero otherwise. For discrete state outputs,

$$
h(Z_t \mid \lambda) = \begin{cases} \frac{1}{c_t}\sum_{i=1}^{N}\left[\gamma_{t|t-1}(i)\nabla_\lambda b_i(o_t) + b_i(o_t) w_t^i\right]; & \text{if } o_t = v_k \\ \frac{1}{c_t}\sum_{i=1}^{N}\left[b_i(o_t) w_t^i\right]; & \text{if } o_t \neq v_k \end{cases}
$$

$$
R_2^i(o_t, \gamma_{t|t-1}, \lambda) = \begin{cases} \sum_{j=1}^{N} a_{ij}\left[1 - \frac{b_i(o_t)\sum_{k=1}^{N}\gamma_{t|t-1}(k)}{c_t}\right]\frac{\gamma_{t|t-1}(i)}{c_t}\nabla_\lambda b_i(o_t), & \text{if } o_t = v_k \\ 0 & \text{if } o_t \neq v_k \end{cases}
$$

For the Gaussian state outputs (2.1):

$$h(Z_t \mid \lambda) = \frac{1}{c_t} \sum_{i=1}^{N} \left[\gamma_{t|t-1}(i) \nabla_{\lambda_b} b_i(o_t) + b_i(o_t) w_t^i \right]$$

$$R_2^i(o_t, \gamma_{t|t-1}, \lambda) = \sum_{j=1}^{N} a_{ij} \left[1 - \frac{b_i(o_t) \sum_{k=1}^{N} \gamma_{t|t-1}(k)}{c_t} \right] \frac{\gamma_{t|t-1}(i)}{c_t} \nabla_{\lambda_b} b_i(o_t)$$

where $\nabla_{\lambda_b} b_i(o_t)$ is given by (2.39).

Collings and Ryden (1998) proposed a similar RMLE approach, for continuous HMM with Gaussian output (2.1). The difference consists of using the sphere parametrization (2.31), instead of the simplex used in (LeGland and Mevel, 1995, 1997). The gradient of the transition probabilities and the recursive computation of the gradient of the state predictive density should be now taken w.r.t. the projected parameters s_{ij} ($a_{ij} = s_{ij}^2$). Although for different purpose, these derivations are very similar to (2.50) and (2.51) presented next.

2.3.3 Minimum Prediction Error (MPE):

Minimizing the output or state prediction error is also considered as alternative objective function. It consists in measuring the error of HMM output prediction (Collings et al., 1994; LeGland and Mevel, 1997) or of HMM filtered state (Ford and Moore, 1998a,b), and then providing an updated estimate of HMM parameters with each new observation symbol.

Recursive Prediction Error (RPE) is first proposed for continuous range Gauss-Markov process (Ljung, 1977), and for a general recursive stochastic gradient algorithm (Arapostathis and Marcus, 1990). RPE extends the concept of least squares from linear to non linear functions. It locates the locale minimum of the prediction error cost function $J(\lambda)$ and provides an updated estimate of the model with each new observation. It is formulated by considering the minimum variance of the prediction error based on the

best model estimate at the time

$$\hat{\lambda} = \arg\min_{\lambda} \left\{ J(\lambda) = \frac{1}{2} E\left[o_t - \hat{o}_t\right]^2 \right\}$$

where $\hat{o}_t$ is the output prediction. The expectation of the off-line minimum variance, for an observation sequence of length T, is approximated by its time average:

$$J_T(\lambda) = \frac{1}{2T} \sum_{t=1}^{T} \left[o_t - \hat{o}_t\right]^2$$

which is commonly minimized with the Gauss-Newton method – a simplified Newton method specialized to the nonlinear least square problem – where each pass k through the data the model is updated using:

$$\hat{\lambda}^{(k+1)} = \hat{\lambda}^{(k)} - \eta_t \left[\frac{1}{T} \sum_{t=1}^{T} \psi_{\lambda^{(k)}} \psi'_{\lambda^{(k)}} \right]^{-1} \left[\frac{1}{T} \sum_{t=1}^{T} \psi_{\lambda^{(k)}} \varepsilon_{\lambda^{(k)}} \right]$$

where $\psi_{\lambda^{(k)}} = \nabla_{\lambda^{(k)}} J_T(\lambda)$ and $\varepsilon_{\lambda^{(k)}} = o_t - \hat{o}_t$. An on-line version of the Gauss-Newton method is also possible if the gradient, and the inverse of the empirical information matrix, can be built up recursively, at each time instant, as new observation arrive. A general practical implementation is given by Ljung and Soderstrom (1983).

Collings et al. (1994) extended the RPE for continuous HMM (known variance is assumed). The model is updated at each time step using:

$$\hat{\lambda}_{t+1} = \mathbb{P}_s \left(\hat{\lambda}_t + \eta_{t+1} R_{t+1}^{-1} \psi'_{t+1} \mathcal{E}_{t+1} \right)$$

where η_t is a gain sequence satisfying (2.44) and R_t is the adaptive matrix which is computed by:

$$R_{t+1} = R_t + \eta_{t+1} (\psi_{t+1} \psi'_{t+1} - R_t)$$

or alternatively, to avoid matrix inversion, R_t^{-1} could be computed as

$$R_{t+1}^{-1} = \frac{1}{1-\eta_{t+1}}\left(R_t^{-1} - \frac{\eta_{t+1}R_t^{-1}\psi_{t+1}\psi_{t+1}'R_t^{-1}}{(1-\eta_{t+1})+\eta_{t+1}\psi_{t+1}'R_t^{-1}\psi_{t+1}}\right)$$

However, now for the HMM case, the output error is given by

$$\mathcal{E}_t = o_t - \hat{o}_{t|t-1}$$

where, the output prediction, $\hat{o}_{t|t-1} = E\left[o_t \mid o_{1:t-1}, \lambda_t\right]$, is conditioned on previous values and given by (using the unnormalized state variable $\alpha_t(i)$ of the Forward-Backward algorithm)

$$\hat{o}_{t|t-1} = \frac{\sum_{i=1}^N \sum_{j=1}^N a_{ij}\alpha_{t-1}(i)b_j(o_t)}{\sum_{i=1}^N \alpha_{t-1}(i)}$$

In addition, a projection on the sphere, $\mathbb{P}_s$, is made at each time step to ensure model constrains (2.31). Accordingly, the gradient of the output error is given by:

$$\psi_t = \left[-\nabla_\lambda \hat{\varepsilon}_t\right]' = \left[\nabla_{\mu_j}\hat{o}_{t|t-1}, \nabla_{s_{ij}}\hat{o}_{t|t-1}\right]'$$

which can be also computed recursively:

$$\begin{aligned}
\nabla_{\mu_j}\hat{o}_{t+1|t} &= \frac{\sum_{i=1}^N a_{ij}\alpha_t(i) + \sum_{i=1}^N a_{ij}\varsigma_t(i)b_j(o_t)}{\sum_{i=1}^N \alpha_{t-1}(i)} - \frac{\sum_{i=1}^N a_{ij}\alpha_t(i)\varsigma_t(i)b_j(o_t)}{\sum_{i=1}^N \alpha_{t-1}^2(i)} \\
\nabla_{s_{ij}}\hat{o}_{t+1|t} &= \frac{2\sum_{i=1}^N s_{ij}\alpha_t(i)\left(\mu_j - \sum_{i=1}^N\sum_{j=1}^N s_{ij}^2 b_j(o_t)\right) + \sum_{i=1}^N a_{ij}\zeta_t(i,j)b_j(o_t)}{\sum_{i=1}^N \alpha_{t-1}^2(i)} \\
&\quad - \frac{\sum_{i=1}^N a_{ij}\alpha_t(i)\zeta_t(i,j)b_j(o_t)}{\sum_{i=1}^N \alpha_{t-1}^2(i)}
\end{aligned} \tag{2.50}$$

where $\varsigma_t(j) = \nabla_{\mu_j}\alpha_t(j)$ and $\zeta_t(j,m,n) = \nabla_{s_{ij}}\alpha_t(j)$ are given by the following recursion:

$$\begin{aligned}
\varsigma_{t+1}(j) &= \sum_{i=1}^N a_{ij}\varsigma_t(j)b_j(o_{t+1}) + \frac{o_{t+1}-b_j(o_t)}{\sigma_w^2}\sum_{i=1}^N a_{ij}\alpha_t(i)b_j(o_{t+1}) \\
\zeta_{t+1}(i,j) &= \sum_{i=1}^N a_{ij}\zeta_t(i,j)b_j(o_{t+1}) - 2s_{ij}\alpha_t(i)\sum_{j=1}^N b_j(o_{t+1})s_{ij}^2 + 2\alpha_t(i)s_{ij}b_j(o_{t+1})
\end{aligned} \tag{2.51}$$

The authors did not study the convergence properties of the algorithm, but through simulations. Later, a convergence problem for this algorithm has been noted when the variance error is small (low noise condition), as explained in (Collings and Ryden, 1998, Sec. 5) and (Ford and Moore, 1998b, Sec. 3-C).

Ford and Moore (1998a) proposed an ELS (extended least squares) and RPE schemes in cases where the transition probabilities are assumed known, which exploit filtered state estimates. These schemes have been extended to the recursive state prediction error (RSPE) algorithm (Ford and Moore, 1998b), where the state transition probabilities are now estimated. Local convergence analysis, for the RSPE, is shown using the ordinary differential equation (ODE) approach developed for the RPE methods. The objective function is given (as a function of the filtered state error) by:

$$\hat{\lambda} = \arg\min_{\lambda} \left\{ J_t(\lambda) = \frac{1}{2} \sum_{t=1}^{\tau} \left[\gamma_{t|t}(i) - \sum_{i=1}^{N} \gamma_{t-1|t-1}(i) a_{ij} \right]^2 \right\}$$

and the model is updated, for each row i, using:

$$\hat{\lambda}_t^i = \hat{\lambda}_{t-1}^i + \left[H_t^i \right]^{-1} w_t^i$$

$$\left[H_t^i \right]^{-1} = \left[H_{t-1}^i \right]^{-1} + \gamma_{t-1|t-1}(i)$$

where H_t^{-1} is an approximation to the second derivative of w_t, and

$$w_t^i = \frac{\partial J_t(\lambda)}{\partial a_{ij}}$$

can be computed recursively, similar to (2.49).

In addition to the RMLE described previously, LeGland and Mevel (1997) proved the convergence of the recursive conditioned least squares estimator (RCLSE), which is a generalization of the RPE approach (Collings et al., 1994). A similar objective function

is used:

$$J_\tau(\lambda) = \frac{1}{2\tau}\sum_{t=1}^{\tau}\left[o_t - \hat{o}_{t|t-1,\lambda}\right]^2$$

where the output prediction given by

$$\hat{o}_{t|t-1} = \sum_{j=1}^{N} b_j(o_t)\gamma_{t|t-1}(j)$$

is based on the state predictive density. Accordingly, its gradients (w_t^i) can be computed recursively using the previously described recursions (2.49) according to each case. The model update for transition probabilities is given, for each row i, by

$$\hat{\lambda}_{t+1}^i = \mathbb{P}_\epsilon\left(\hat{\lambda}_t^i + \eta_{t+1}\left[\mu_i(o_t - \sum_{j=1}^{N}\mu_j\gamma_{t|t-1}(j))\right]w_t^i\right)$$

and for the continuous state outputs by

$$\hat{\lambda}_{t+1}^i = \mathbb{P}_\epsilon\left(\hat{\lambda}_t^i + \eta_{t+1}\left(\sum_{i=1}^{N}\left[\mu_i(o_t - \sum_{j=1}^{N}\mu_j\gamma_{t|t-1}(j))\right]w_t^i + \sum_{i=1}^{N}\left[\nabla_i\mu_i(o_t - \sum_{j=1}^{N}\mu_j\gamma_{t|t-1}(j))\right]\gamma_{t|t-1}(i)\right)\right)$$

As for the discrete state outputs, the algorithm is only applicable for finite alphabet. In this case, the same formulas apply, however with $\mu_i = \sum_{t=1}^{T} o_t b_i(o_t)$ and of course using the gradient recursion (2.49) for the discrete case.

2.4 An Analysis of the On-line Learning Algorithms

Algorithms for on-line learning of HMM parameters are mostly designed to learn observations from a source generating an infinite amount of data. For instance, these algorithms can be employed to re-estimate HMM parameters from a large number of sub-sequences (e.g., speech sentences), or a stream of symbols (e.g., signal in a noisy communication channel). Block-wise techniques re-estimate HMM parameters after observing each sub-sequence, while symbol-wise techniques re-estimate HMM parameters

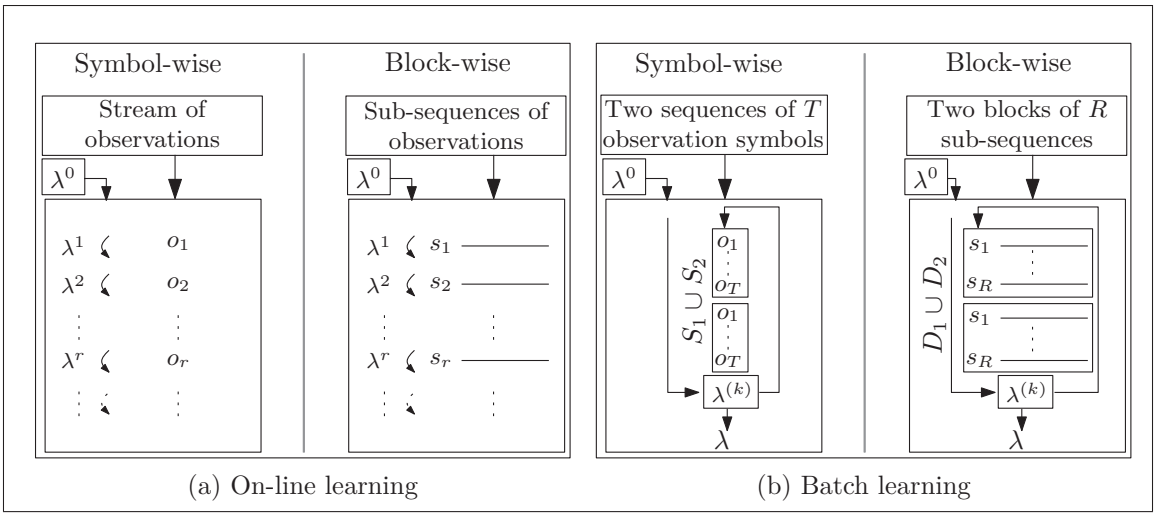


Figure 2.5 An illustration of on-line learning (a) from an infinite stream of observation symbols (o_i) or sub-sequences (s_i) versus batch learning (b) from accumulated sequences of observations symbols ($S_1 \cup S_2$) or blocks of sub-sequences ($D_1 \cup D2$)

upon observing each new symbol (Figure 2.5 (a)). The objective is to optimize HMM parameters to fit the source generating the data through one observation of the data. In contrast, batch learning algorithms re-estimate HMM parameters upon observing all accumulated sequences or blocks of sub-sequences of observations (Figure 2.5 (b)). Batch learning algorithms optimize HMM parameters to fit the accumulated data over several iterations.

On-line learning is challenging since algorithms must converge rapidly to the true model with minimum resource requirements. It is assumed that the true generative model resides within the HMM parameters space. Finding recursive formulas to update the HMM parameter is necessary for on-line learning but not sufficient. Statistical properties such as consistency and asymptotic normality must be proven to ensure the convergence of an on-line learning algorithm. This section provides an analysis of the convergence properties and of the computational complexity of the on-line algorithms presented in Section 2.3.

2.4.1 Convergence Properties

The extension of the batch EM to an incremental EM version using a partial data for the E-step followed by a direct update of the model parameters in the M-step, is a well known technique to overcome the resource requirements when processing a large fixed-size data set (Hathaway, 1986). Neal and Hinton (1998) showed that this EM variant gives non-increasing divergence, and that local minima in divergence are local maxima in likelihood. However, as argued by Gunawardana and Byrne (2005), this is insufficient to conclude that it converges to local maxima in the likelihood. They showed that this incremental EM variant is not in fact an EM procedure and hence the standard GEM convergence results (Wu, 1983) do not apply. This is because the E-step is modified to use summary statistics from the posterior distributions of previous estimates. By using the Generalized Alternating Minimization (GAM) procedure in an information geometric framework, Gunawardana and Byrne (2005) provided a convergence analysis which proves that the incremental EM converges to stationary points in likelihood, although *not monotonically*.

The authors could find no proof of convergence for on-line EM-based algorithms when HMM parameters estimation is performed from infinite amount of data, for both block-wise (Digalakis, 1999; Mizuno et al., 2000) and symbol-wise (Cappe, 2009; Florez-Larrahondo et al., 2005; Mongillo and Deneve, 2008; Stenger et al., 2001; Stiller and Radons, 1999) techniques. Furthermore, statistical analysis such as consistency and asymptotic normality are not provided. Although the rate of convergence is not studied analytically, applying stochastic techniques in literature such as averaging (Polyak, 1991) has been shown to help improve convergence properties (Cappe, 2009). Among the block-wise algorithms, the gradient-based algorithm proposed by Cappe et al. (1998) should achieve the fastest convergence rate since it is based on a quasi-Newton method (second order approximation), while the others should have similar slower convergence rate. Similarly, for the recursive EM symbol-wise algorithms (Krishnamurthy and Moore, 1993; Slingsby,

1992) since the complete likelihood is optimized by a second order method, although no proof of convergence is provided.

Among the methods based on RMLE, Ryden (1998, 1997) provided convergence analysis for his block-wise RMLE. He proved the consistency by using the classical result of Kushner and Clark (1978). Independently, LeGland and Mevel (1995, 1997) proved the convergence and asymptotic normality of their symbol-wise RMLE, under the assumption that the transition probability matrix is primitive, yet without any stationary assumption. This is accomplished by using the geometric ergodicity and the exponential forgetting of the predictive filter and its gradient. Krishnamurthy and Yin (2002) extended the RMLE results of LeGland and Mevel (1997) to auto-regressive models with Markov regime, and added results on convergence, rate of convergence, model averaging, and parameter tracking. In particular, they showed that the above assumption can be relaxed to the condition in which the transition probability matrix is aperiodic and irreducible. The authors also suggested using the iterate averaging (Polyak, 1991), as well as observation averaging for faster convergence and lower variances.

RPE based techniques were first extended and applied for recursive estimation of HMM parameters due to their quadratic convergence rate (although asymptotically sub-optimal). This convergence speed comes at the expenses of an increased complexity, and as discovered later, the RPE algorithm proposed by Collings et al. (1994) suffers from numerical issues. Local convergence analysis is only shown for the RSPE (Ford and Moore, 1998a) using the ordinary differential equation. Finally, similar to the RMLE, LeGland and Mevel (1997) also proved the convergence of the RCLSE.

2.4.2 Time and Memory Complexity

The time complexity of an algorithm, can be analytically defined as the sum of the worst-case running times $Time_{\{p\}}$ for each operation p required to re-estimate HMM parameters

Table 2.1 Time and memory complexity for some representative block-wise algorithms used for on-line learning of a new sub-sequence $o_{1:T}$ of length T. N is the number of HMM states and M is the size of alphabet symbols

Algo.	**Step**	**# Multiplications**	**# Divisions**	**# Exp**	**Memory**
Baldi and Chauvin (1998)	γ and ξ	$6N^2T+3NT-6N^2-N$	$N^2T+3NT-N^2-N$		$NT+N^2+2N$
	A	$2N^2$	N^2	N^2	$2N^2$
	B	$2NM$	MN	NM	$2NM$
	Total	$6N^2T+3NT-4N^2$ $+2NM-N$	$N^2T+3NT+MN-N$	N^2+NM	$NT+3N^2+2NM+2N$
Cappe et al. (1998), quasi-Newton based	$\nabla\ell$and γ_t	$5N^2T+2NT$	$3N^2T+NT$		$N(N^2+NM)$
	A	$2N^2$	N^2		$2N^2$
	B	$2N$	N		$2N$
	Total	$5N^2T+2NT+2N^2$ $+2N$	$4N^2T+NT+2N$ $+N$		$2N^3+2N^2$
Mizuno et al. (2000)	γ and ξ	$6N^2T+3NT-6N^2-N$	$N^2T+3NT-N^2-N$		$NT+N^2+2N$
	A	N^2	N^2		N^2
	B	NM	NM		NM
	Total	$6N^2T+3NT-5N^2$ $+NM-N$	N^2T+3NT $+NM-N$		$NT+2N^2+NM+2N$

based on new training sequence (block-wise) or new observation (symbol-wise):

$$Time = \sum_p Time_{\{p\}} = \sum_p t_p n_p$$

where t_p is the constant time needed to execute an operation p (e.g., multiplication, division, etc.), and n_p is the number of times this operation is executed. The *growth rate* is then obtained by making the key parameters (T and N) of the worst-case time complexity tend to ∞. For simplicity, only most costly type of operation is considered – multiplication, division, and exponent. Time complexity is independent from the convergence time, which varies among different algorithms as discussed in Section 2.4.1. Memory complexity is estimated as the number of 32 bit registers needed during learning process to store temporary variables. The worst-case memory space required for estimation of sufficient statistics such as conditional distributions of states and gradients of the log-likelihood is considered.

Tables 2.1 and 2.2 present a breakdown of time and memory complexity for some representative algorithms. Table 2.1 compares the complexity of block-wise algorithms applied to on-line learning of an observation sequence $o_{1:T}$ of length T and then updating the model parameters. Block-wise techniques typically employ a fixed-lag smoothing approach for estimating HMM states upon receiving each sub-sequence. In general, all these algorithms have the same time complexity, $\mathcal{O}(N^2T)$. However, EM-based block-wise techniques (Digalakis, 1999; Mizuno et al., 2000) are easier to implement than gradient-based (Baldi and Chauvin, 1994; Cappe et al., 1998; Singer and Warmuth, 1996) and RMLE (Ryden, 1998, 1997) techniques. EM-based techniques maintain the parameters constraints (2.2) and (2.3) implicitly, and do not employ derivatives and projections of the gradient filter. Accordingly, they require fewer computations to update the HMM parameters upon receiving an observation symbol (see Table 2.1). Although, block-wise algorithms have the same memory complexity of $\mathcal{O}(NT)$, the memory requirements of the algorithm proposed by Cappe et al. (1998) is independent of the sequence length T. This is due to the recursion on the gradient itself and could be useful when T is large. However, it requires additional time complexity due to an internally employed line search technique.

Table 2.2 compares the complexity of symbol-wise algorithms for on-line learning from a stream of observation symbols o_t. The time complexity is based on the learning of one observation. Therefore, it must be multiplied by the length, T, of a finite observation sequence for comparison with block-wise techniques. Algorithms that requires more than N^2 computational complexity per time step may become less attractive when the number of HMM states N is large. As for block-wise techniques, EM-based symbol-wise filtering and prediction techniques (Florez-Larrahondo et al., 2005; Stenger et al., 2001) are generally easier to implement than gradient-based (Collings and Ryden, 1998; Garg and Warmuth, 2003; LeGland and Mevel, 1995, 1997) and minimum prediction error (Collings et al., 1994; Ford and Moore, 1998a,b) techniques. On the other hand, although EM-based smoothing techniques (Cappe, 2009; Mongillo and Deneve, 2008; Stiller and Radons, 1999) have been shown to provide more accurate states estimate

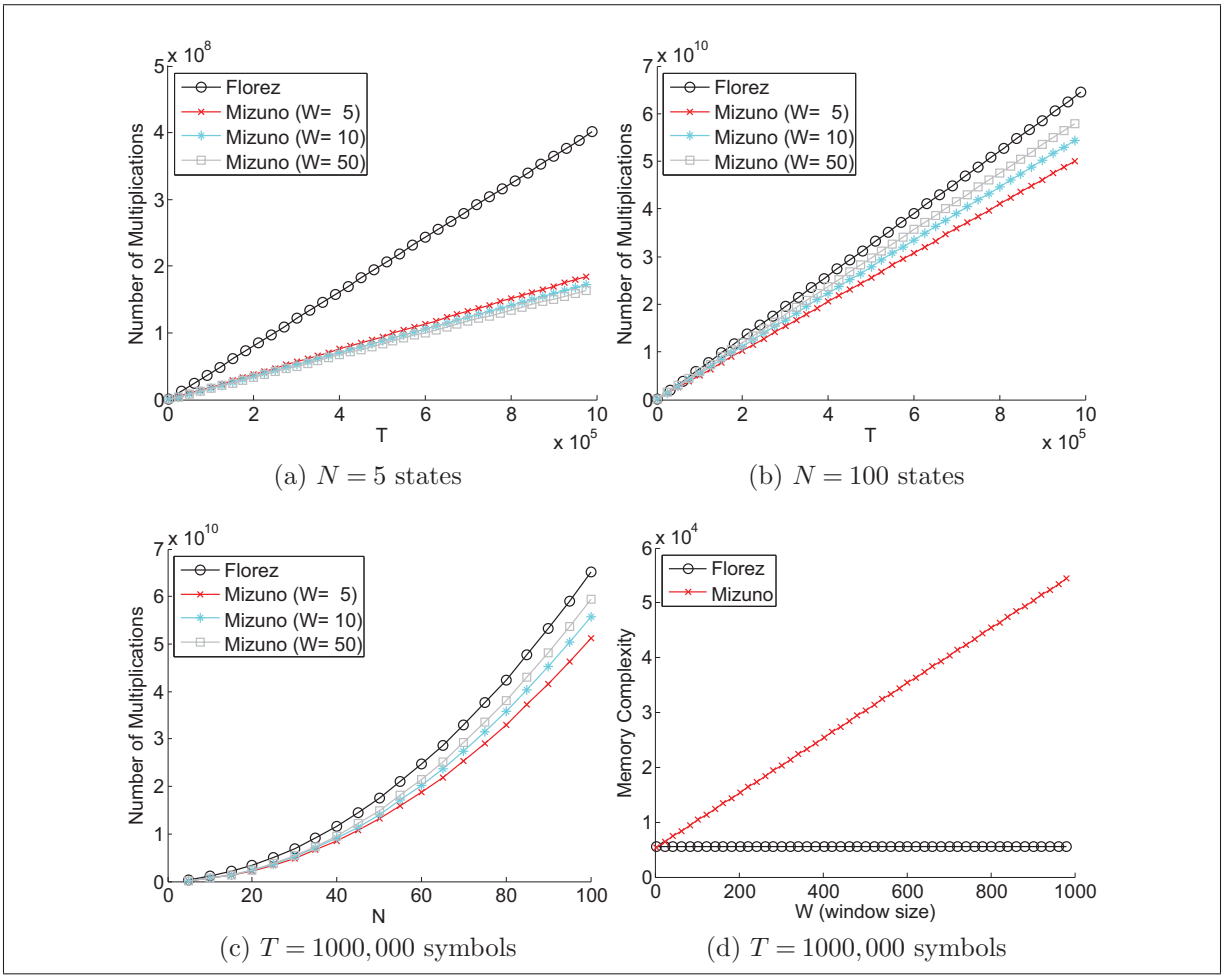
Table 2.2 Time and memory complexity of some representative symbol-wise algorithms used for on-line learning of new symbol o_i. N is the number of HMM states and M is the size of alphabet symbols

Algorithms	**Step**	**# Multiplications**	**# Divisions**	**Memory**
Stiller and Radons (1999), smoothing-based approach	$\xi^t_{ijk}(o_t)$	N^4+N^2	$2N^3$	$2N^3M$
	$A=\{a_{ij}(o_t)\}$	–	N^2M	N^2M
	Total	N^4+N^2	$2N^3+N^2M$	$2N^3M+N^2M$
Florez-Larrahondo et al. (2005), prediction-based approach	γ and ξ	$5N^2+N$	N^2+N	N^2+3N
	A	N^2	$2N^2$	N^2
	B	NM	$2NM$	NM
	Total	$6N^2+NM+N$	$3N^2+2NM+N$	$2N^2+NM+3N$
Slingsby (1992)	d, g,γ	$6N^2+N$	$2N^2+N$	$2N^2+N$
	A	N^2	N^2+2N	N^2
	B	$NM+3N$	$NM+2N$	NM
	Total	$7N^2+NM+4N$	$3N^2+NM+5N$	$3N^2+NM+N$
Legland and Mevel (1995)	R_1 and R_2	N^3+N^2M	N^2+NM	$N^2+NM+4N$
	A	N^2	$2N^2$	N^2
	B	NM	$2NM$	NM
	Total	$N^3+N^2(M+1)+NM$	$3N^2+3NM$	$2N^2+2NM+4N$
Collings et al. (1994)	$R_t^{-1},\psi_t,\mathcal{E}_t$	$N^4+4N^3+N^3+5N^2+5M^2+3NM$	$3N^2+M^2+M$	$N^2+NM+2N$
	A	– (only N^2 additions) –		N^2
	B	– (only NM additions) –		NM
	Total	$N^4+4N^3+N^3+5N^2+5M^2+3NM$	$3N^2+M^2+M$	$2N^2+2NM+4N$

than filtering and prediction (Anderson, 1999), their time complexity per observation is $\mathcal{O}(N^4)$, which makes them less attractive when the number of HMM states N is large. In general the memory requirement of symbol-wise techniques is independent of T, since the HMM re-estimation is either based the current symbol (filtering) or on small finite predictions – one symbol look-ahead or a larger fixed-lag smoothing.

Block-wise algorithms typically require fewer computations than symbol-wise algorithms when learning from a large observation sequence, at the expenses of an increased memory requirements. This is mainly due to fewer update of HMM parameters that is required with block-wise learning algorithms. Figure 2.6 illustrates the time and memory complexity required by the EM-based symbol-wise prediction algorithm proposed by Florez-

Larrahondo et al. (2005) versus its block-wise fixed-lag smoothing counterpart proposed by Mizuno et al. (2000), when learning an observation sequence of length $T = 1000,000$ symbols, with an output alphabet of size $M = 50$ symbols. While symbol-wise algorithms can be directly employed to learn such sequence of observations, block-wise algorithms require segmenting the sequence into non-overlapping sub-sequences using a sliding window of size W, which is also the fixed-lag value.



(a) $N = 5$ states (b) $N = 100$ states

(c) $T = 1000,000$ symbols (d) $T = 1000,000$ symbols

Figure 2.6 An example of the time and memory complexity of a block-wise (Mizuno et al. (2000)) versus symbol-wise (Florez-Larrahondo et al. (2005)) algorithm for learning an observation sequence of length $T = 1000,000$ with an output alphabet of size $M = 50$ symbols

As shown in Figure 2.6 (a), (b) and (c), block-wise algorithms are more time efficient with smaller number of states N and smaller window sizes W. While symbol-wise algo-

rithms require T updates of HMM parameters, block-wise algorithms require about $\frac{T}{W}$ updates when learning the same observation sequence. When the N value increases the computational time of block-wise algorithms may approach that of symbol-wise for large window sizes. In such cases, the computational time of parameters update is dominated by that of state densities which scales quadratically with N and linearly with W. Therefore, for a given N value, smaller window sizes should be favored since they reduce the time complexity for block-wise learning from a large observation sequence. The memory complexity is also decreased linearly with the window size as shown in Figure 2.6 (d). Increasing the fixed-lag value W may yield a more accurate local estimate of HMM states, and hence improves algorithms stability. If memory constraints are an issue, block-wise techniques that employ recursions on the gradient itself (Cappe et al., 1998) should be favored since their memory complexity are independent of W. When the application imposes stricter constraints on the memory space, symbol-wise algorithms should be employed. The strategy for selecting a value for W is thereby an additional issue to address with block-wise techniques.

2.5 Guidelines for Incremental Learning of HMM Parameters

The target application implies a scenario for data organization and learning. Depending on the application, incremental learning may be performed on an abundant or limited amount of new data. In addition, this data may be organized into a block of observation sub-sequences or into one long observation sequence. Within the incremental learning scenario (Figure 2.1), HMM parameters should be re-estimated using the newly-acquired training data, without corrupting the previously acquired knowledge, and therefore degrading the system performance. One of the key issues is optimizing HMM parameters such that contributions of new data and pre-existing knowledge is balanced. The choice of user-defined parameters (e.g., learning rate and sub-sequence length) have a significant impact on HMM performance. The rest of this section provides guidelines and underscores challenges faced when applying on-line techniques to supervised incremental learning of new training data, when the data is either abundant or limited.

2.5.1 Abundant Data Scenario

Under the abundant data scenario, it is assumed that a long sequence of training observations becomes available to update HMM parameters through incremental learning. A more complete view of phenomena is therefore presented to the learning algorithm. As stated previously, symbol-wise algorithms can be directly employed to learn such sequences, one observation at a time, while block-wise algorithms require buffering some amount of data using for instance a sliding window according to a user-defined buffer size W.

When learning is performed in a static environment from a large sequence of observations, one view of the data is typically sufficient to capture the underlying structure by exploiting patterns redundancy. This is because the abundant data provide a more complete view of phenomena. Resource constraints would be another reason behind restricting the iterative learning procedure to one pass. Therefore, the convergence properties of an on-line algorithm must be known to determine the speed and limits of convergence behavior. As described in Section 2.4.1, when the true model is contained in the set of solutions, some algorithms are shown to be consistent and asymptotically normal by properly choosing a monotonically decreasing learning rate η_t and by applying Polyak-Ruppert averaging (Polyak, 1991). For on-line learning in static environment, decreasing step-sizes are essential conditions of convergence. Fixed step-sizes may yield the convergence of algorithms to oscillate around their limiting values with variance proportional to the step-size. However, some novelty criteria on the new data should be employed to reset the monotonically decreased learning rate when provided with a new sequence of observations.

In contrast, when learning is performed in dynamically-changing environments, the notion of optimality is no longer valid because the on-line algorithm should forget past knowledge and adapt rapidly to the newly acquired information. Tracking non-stationary environments and handling slow drift is typically achieved by choosing a fixed learn-

ing rate η. For abrupt drift however, a data driven learning rate is required to detect changes and adapt the step-sizes according to the incoming data (Schraudolph, 1999; Stiller, 2003).

In both static and dynamically-changing environments, algorithms with low time and memory complexity are favored to learn the long sequence of observations. As discussed in Section 2.4.2, symbol-wise algorithms that requires more than N^2 time complexity upon receiving each symbol are less attractive, especially when the number of HMM states N is large. For block-wise algorithms, smaller window sizes W are favored for reducing both time and memory complexity. However, the stability of algorithms must also be considered when selecting the window size.

Some issues that require further investigation when adapting the HMM parameters from abundant data. Empirical benchmarking studies of these techniques could offer further insight on trade-offs for selecting algorithms and user-defined parameters for an application domain. For instance, the performance of techniques based on MLE (such as EM and gradient based) should be compared versus those based on MMD, and recursive techniques for MPE. An interesting comparison would also involve symbol-wise versus block-wise and filtering versus fixed-lag smoothing. Significant improvement, accuracy, speed and stability of convergence, computing resources, and amount of training data required to reach or maintain a given level of performance should be among the evaluation criteria. To this end, it is recommended to conduct non-parametric tests for statistical comparisons of multiple classifiers over multiple data sets (Demšar, 2006; García and Herrera, 2008). In addition to assessing the strength and weakness of each algorithm, such comparison may lead to efficient hybrid on-line algorithms that require fewer resources (see Section 2.2.2.3).

2.5.2 Limited Data Scenario

Under the limited data scenario, it is assumed that a short sequence of observations becomes available to update the HMM parameters, providing therefore a limited view of

phenomena. For block-wise algorithms the sequence is typically segmented into shorter sub-sequences using a sliding window with a user-defined window size W. In contrast to the abundant data case, when provided with limited data, several iterations over the new sequence or block of observations are required to re-estimate HMM parameters. This local optimization raises additional challenges. Specialized strategies are needed for managing the learning rate which, at each iteration must balance integration of pre-existing knowledge of the HMM and the newly-acquired training data. In addition, the learning technique may become trapped in local maxima on the cost function on the parameter space.

To illustrate the difficulty, assume that the training block D_2, which contains one or multiple sub-sequences, becomes available after a HMM has been previously trained on a block D_1 and deployed for operations. After training on D_1, the HMM has a parameter settings of λ_1. An on-line algorithm that optimizes, for instance, the log-likelihood function requires re-estimating the HMM parameters over several iterations until this function is maximized for D_2. The plain curve in Figure 2.7 represents the log-likelihood function of an HMM(λ_1) that was previously learned D_1, and then has incrementally learned D_2 over the space of on HMM parameter (λ). Since λ_1 is the only information at hand from previous data D_1, the training process starts from HMM(λ_1). Optimization of HMM parameters depends on the shape and position of the log-likelihood function of HMM(λ_2) with respect to that of HMM(λ_1). For instance, if λ_1 was selected according to point (a) in Figure 2.7, then the optimization will probably lead to point (d), which degrades HMM performance. If λ_1 was selected as point (b), the optimization would probably lead to a better solution (f). In contrast, training a new HMM(λ_2) on D_2 by starting from the start on different initializations may lead to point (g) which, depending on the new data, may yield higher log-likelihood than all previous models. Combinations of HMM(λ_1) and HMM(λ_2) at the classifier level may however provide improved performance in such cases.

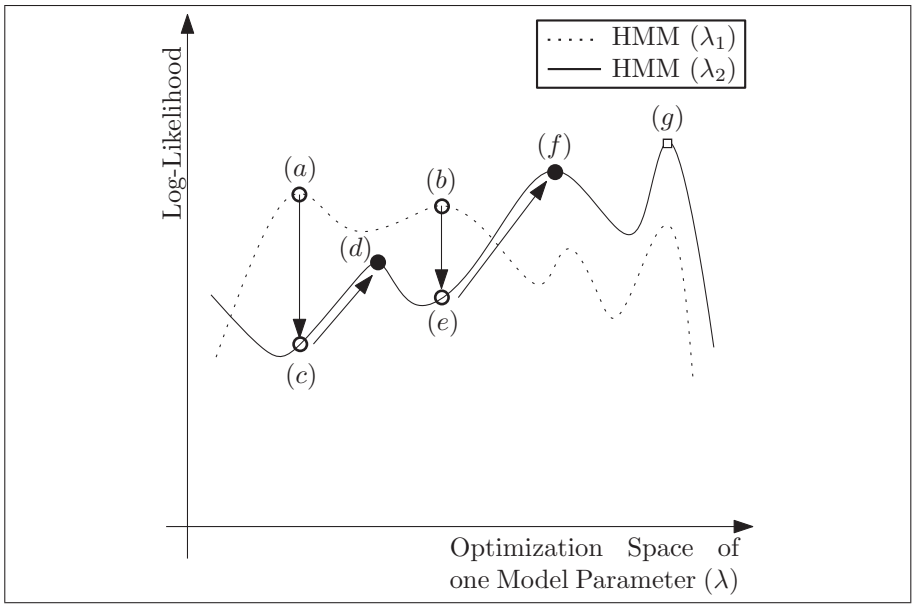


Figure 2.7 The dotted curve represents the log-likelihood function associated with and HMM (λ_1) trained on block D_1 over the space of one model parameter λ. After training on D_1, the on-line algorithm estimates $\lambda = \lambda_1$ (point (a)) and provides HMM (λ_1). The plain curve represents the log-likelihood function associated with HMM (λ_2) trained on block D_2 over the same optimization space

Typical procedures for unbiased performance estimation should be employed. This involves stopping the training iterations when the log-likelihood of an HMM on independent validation data no longer improves. Using the hold-out or cross-validation procedures reduces the effects of overfitting. HMM parameters that yield the maximum log-likelihood value on other validation data are then selected, which guarantees generalization performance during operations.

When the situation allows, some proportion of a new training data should be dedicated to validation. This would provide a better the stopping criterion and improve the generalization capabilities of the classification system. In order to perform such validation, a set of observations must be stored and updated over time in a fixed-sized buffer. Managing this validation set over time depends on the application environment and should be selected and maintained according to some relevant selection criteria (e.g., some information-theoretic criteria). For instance, in dynamically-changing environments, older validation

data should be discarded and replaced with new observations data that is more representative of the underlying data distribution. In static or cycle-stationary environments, older data should be discarded in a first-in-first-out manner.

A potential advantage of applying the on-line learning techniques over batch learning techniques to limited data scenario resides in their added stochasticity, which stems from the rapid re-estimation of HMM parameters. This may aid escaping local maxima during the early adaptation to newly provided data. Managing the internal learning rate, associated with each sub-sequence or observation symbol, is therefore different for limited data than for abundant data. In general, employing a fixed learning rate along with the validation strategies described above would maintaining the level of performance in both static and dynamically-changing environments. However, more insights are required in this regard to determine if applying a monotonically decreasing or auto-adaptive learning rates at the iteration, sequence, or symbol levels are beneficial for escaping local maxima and hence improving the performance.

The resource requirements for the on-line algorithms when learning from limited data are comparable to that of when learning from abundant data scenario and presented in Section 2.4.2 except for abundant data T is much bigger. The only difference is that the complexity presented for both block and symbol wise techniques must be multiplied by the number of training iterations, which varies according to the algorithm employed, validation strategy and stopping criteria, and training data. Learning from limited data is less restrictive to memory and time complexity than learning from abundant data. Therefore, even the most compute-intensive symbol-wise algorithms – requiring $\mathcal{O}(N^4)$ time complexity upon receiving each symbol – or block-wise algorithms – requiring $\mathcal{O}(N^4T)$ time and $\mathcal{O}(NT)$ memory complexity upon receiving each sub-sequence – can be afforded as long as they improve or maintain the HMM performance. In limited data scenario, escaping local maxima, managing learning rates and stopping criteria may be critical factors for selecting an on-line algorithm than minimizing the resource requirements. Finally, as for the abundant data scenario, applying the on-line learning techniques from

this survey to adapt HMM parameters to limited new data requires comparative benchmarking and statistical testing at the objective functions, optimization techniques and algorithms (symbol-wise versus block-wise) levels, and according to various user-defined parameters (learning rate and window size).

2.6 Conclusion

This chapter present a survey of techniques found in literature that are suitable for incremental learning of HMM parameters. These include on-line learning algorithms that have been initially proposed for learning from a long training sequence, or block of sub-sequences. In this chapter, these techniques are categorized according to the objective functions, optimization techniques and target application. Some techniques are designed to update HMM parameters upon receiving a new symbol (symbol-wise), while others update the parameters upon receiving a sequence (block-wise). Convergence properties of these techniques are presented, along with an analysis of their time and memory complexity. In addition, the challenges faced when these techniques are applied to incremental learning is assessed for scenarios in which the new training data is limited and abundant.

When the new training data is abundant (scenario 1), the HMM parameters should be re-estimated to fit the source generating the data through one pass over a sequence of observations. When new data corresponds to a long sequence or block of sub-sequences generated from a stationary source (static environment), these techniques must employ a specialized strategy to manage the learning rate such that new data and existing knowledge are integrated without compromising the HMM performance. This includes resetting a monotonically decreasing learning rate when provided with new data or employing an auto-adaptive learning rate. Rapid convergence with limited resource requirements are other major factors in such cases. However few learning algorithms have been provided with a proof of convergence or other relevant statistical properties. Another important issue is the ability to operate in dynamically-changing environments. In such cases, some

novelty criteria on the new data should be employed to detect changes and trigger adaptation by resting a monotonically decreasing learning rate or fine-tuning an auto-adaptive learning rate.

When the new training data is limited (scenario 2), HMM parameters should be re-estimated over several iterations due to the limited view of phenomena. Therefore, HMM parameters should be able to escape local maxima of the cost function associated with the new data. Given the limited data, early stopping criteria through hold-out or cross validation must be considered when learning the new block of data to reduce the effects of overfitting. Accumulating and updating representative validation data set over time must be considered. However, this requires investigating some selection criteria for maintaining the most informative sequences of observations and discarding less relevant ones. Another major challenge resides in adapting the current operational HMM to the newly-acquired data without corrupting existing knowledge. One possible solution involves using an adaptive learning rate which, at each iteration, controls the weight given to the current HMM with reference to the new information. This learning rate should preferably be inferred from the data.

Finally, this chapter underscores the need for comparative benchmarking studies of these on-line algorithms for incremental learning in both abundant and limited data scenarios. Some interesting comparative studies, evaluation criteria and statistical testing methods are suggested for selecting an algorithm for a particular situation. Knowledge corruption and performance degradation may arise because incremental learning of new data is initiated from a pre-existing HMM. One promising solution involves combining HMMs at either classification or response levels. In such cases, new HMMs would be trained using the newly-acquired data, and combined with those trained on the previously-acquired data.

Next chapter describes the novel decision-level Boolean combination technique proposed for efficient fusion of the responses from multiple classifiers in the ROC space.

CHAPTER 3

ITERATIVE BOOLEAN COMBINATION OF CLASSIFIERS IN THE ROC SPACE: AN APPLICATION TO ANOMALY DETECTION WITH HMMS*

In this chapter, a novel Iterative Boolean Combination (*IBC*) technique is proposed for efficient fusion of the responses from multiple classifiers in the ROC space. It applies all Boolean functions to combine the ROC curves corresponding to multiple classifiers, requires no prior assumptions, and its time complexity is linear with the number of classifiers. The results of computer simulations conducted on both synthetic and real-world Host-Based intrusion detection data indicate that the *IBC* of responses from multiple HMMs can achieve a significantly higher level of performance than the Boolean conjunction and disjunction combinations, especially when training data is limited and imbalanced. The proposed *IBC* is general in that it can be employed to combine diverse responses of any crisp or soft one- or two-class classifiers, and for wide range of application domains.

3.1 Introduction

Intrusion Detection Systems (IDS) is used to identify, assess, and report unauthorized computer or network activities. Host-Based IDSs (HIDS) are designed to monitor the activities of a host system and state, while network-based IDSs (NIDS) monitor the network traffic for multiple hosts. HIDSs and NIDSs have been designed to perform misuse detection and anomaly detection. Anomaly-based intrusion detection allows to detect novel attacks for which the signatures have not yet been extracted (Chandola et al., 2009a). In practice, anomaly detectors will typically generate false alarms due in large part to the limited data used for training, and to the complexity of underlying data

*THIS CHAPTER IS PUBLISHED AS AN ARTICLE IN PATTERN RECOGNITION JOURNAL, DOI: 10.1016/J.PATCOG.2010.03.006

distributions that may change dynamically over time. Since it is very difficult to collect and label representative data to design and validate an Anomaly Detection Systems (ADS), its internal model of normal behavior will tend to diverge from the underlying data distribution.

In HIDSs applied to anomaly detection, operating system events are usually monitored. Since system calls are the gateway between user and kernel mode, traditional host-based anomaly detection systems monitor deviation in system call sequences. Forrest et al. (1996) confirmed that short sequences of system calls are consistent with normal operation, and unusual burst will occur during an attack. Their anomaly detection system, called Sequence Time-Delay Embedding (STIDE), is based on look-up tables of memorized normal sequences. During operations, STIDE must compare each input sequence to all "normal" training sequences. The number of comparisons increases exponentially with the detector window size. Moreover, STIDE is often used for design and validation of other state-of-the-art detectors. Various neural and statistical detectors have been applied to learn the normal process behavior through system call sequences (Warrender et al., 1999). Among these, techniques based on discrete Hidden Markov Models (HMMs) (Rabiner, 1989) have been shown to produce a very high level of performance (Warrender et al., 1999). A well trained HMM is able to capture the underlying structure of the monitored application and detect deviations from "normal" system call sequences. Once trained, an HMM provides a fast and compact detector, with tolerance to noise and uncertainty.

Designing an HMM for anomaly detection involves estimating HMM parameters and order (number of hidden states, N) from the training data. The value of N has a considerable impact not only on the detection rate but also on the training time. In the literature on HMMs applied to anomaly detection (Gao et al., 2002; Hoang and Hu, 2004; Warrender et al., 1999), the number of states is often chosen heuristically or em-

pirically using validation data[1]. In addition, HMMs are designed to provide accurate results for a particular window size and anomaly size. Therefore, a single best HMM will not provide a high level of performance over the entire detection space. In a previous work (Khreich et al., 2009b), the authors proposed a multiple-HMM (μ-HMM) approach, where each HMM is trained using a different number of hidden states. During operations, each HMM outputs a probability that the HMM produced the input sequence. HMM responses are combined in the Receiver Operating Characteristics (ROC) space according to the Maximum Realizable ROC (MRROC) technique (Scott et al., 1998). This technique is robust in imprecise environments where for instance prior probabilities and/or classification costs may change (Provost and Fawcett, 1997), and can be always applied in practice without any assumption of independence between detectors (Scott et al., 1998). Results have shown that this μ-HMMs approach can provide a significant increase in performance over a single best HMM and STIDE (Khreich et al., 2009b).

Boolean functions – especially the conjunction AND and disjunction OR operations – have recently been investigated to combine crisp or soft detectors within the ROC space. Successful applications for such combination include machine learning (Barreno et al., 2008; Fawcett, 2004), biometrics (Tao and Veldhuis, 2008), bio-informatics (Haker et al., 2005; Langdon and Buxton, 2001), automatic target recognition (Hill et al., 2003; Oxley et al., 2007). Boolean Combination (BC) based on conjunction or disjunction has been shown to improve performance over the MRROC in many applications, yet requires idealistic assumptions in which the detectors are conditionally independent and their respective ROC curves are smooth and proper. In contrast, applying all Boolean functions, using an exhaustive brute-force search to determine optimal combinations leads to an exponential explosion of combinations, which is prohibitive even for a small number of crisp detectors (Barreno et al., 2008).

[1]This is also the case in many other applications of ergodic HMMs. However, recently nonparametric Bayesian approaches have been proposed to overcome HMMs order selection (Beal et al., 2002; Gael et al., 2008).

In practice, neural and statistical classifiers are typically trained with limited amount of representative data, and class distributions are often complex and imbalanced, which leads to concavities on empirical ROC curves, and to poor performance. This issue is especially critical in binary classification problems, such as anomaly detection, where samples from the positive class[2] are inherently rare, and are costly to analyze. In such cases the ROC curve typically comprises large concavities. Some authors have argued that ROC curves can be simply repaired using the ROCCH prior to the conjunction or disjunction combinations (Haker et al., 2005). However this technique is inefficient, since the ROCCH selects thresholds of the locally superior points and discard the rest. This leads to a loss of diverse information which could be used to improve performance.

In this chapter, the problem of ROC-based combination is addressed in the general case, where detectors are trained with limited and imbalanced training data. An Iterative Boolean Combination (IBC_{ALL})[3] technique is proposed to efficiently combine the responses from multiple detectors in the ROC space. In contrast with most techniques in literature, where only the AND or OR are investigated, the IBC_{ALL} exploits *all* Boolean functions applied to the ROC curves, and it does not require any prior assumption regarding the independence of the detectors and the convexity of ROC curves. At each iteration, the IBC_{ALL} selects the combinations that improve the ROCCH and recombines them with the original ROC curves until the ROCCH ceases to improve. Although it seeks a sub-optimal set of combinations, the IBC_{ALL} is very efficient in practice and does not suffer from the exponential explosion (Barreno et al., 2008), and it provides a higher level of performance than related techniques in literature (Haker et al., 2005; Scott et al., 1998; Tao and Veldhuis, 2008). The IBC_{ALL} technique can be applied when training data is limited and test data is heavily imbalanced. Another advantage of the proposed technique is its ability to repair the concavities in the ROC curve when applied to combine the responses of the same ROC curve. The IBC_{ALL} is general in the sense

[2]The positive or target class is typically the class of interest for the detection problem. For anomaly detection, the target class corresponds to the intrusive or abnormal class.

[3]The subscript "ALL" is used to emphasize that the IBC technique employs all Boolean functions.

that can be applied to combine the responses of any soft, crisp, or hybrid detector in the ROC space, whether the corresponding curves result from the same detector trained on different data or trained according to different parameters, or from different detectors trained on the same data.

During computer simulations, multiple HMMs are applied to anomaly detection based on system calls. The performance obtained by combining the responses of μ-HMMs in ROC space with the proposed IBC_{ALL} technique is compared to that of MRROC fusion (Scott et al., 1998), to that of the conjunction (BC_{AND}) and disjunction (BC_{OR}) combinations (Haker et al., 2005; Tao and Veldhuis, 2008), and to that of STIDE (for reference). To investigate the effect of repairing the concavities of the ROC curves on the combinations, each ROC curve is first repaired using the IBC_{ALL} technique, and then all curves are combined with the MRROC. This is also compared to the Largest Concavity Repair (LCR) proposed in (Flach and Wu, 2005). The impact on performance of using different training set sizes, detector window sizes and anomaly sizes is assessed through the area under the curve (AUC) (Hanley and McNeil, 1982), partial AUC (Walter, 2005), and true positive rate (tpr) at a fixed false positive rate (fpr). The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets.

The rest of this chapter is organized as follows. The next section describes the application of discrete HMMs in anomaly-based HIDS. In Section 3.3, existing techniques for combination of detectors in the ROC space are presented. The proposed technique for Iterative Boolean Combination of detector responses is presented in Section 3.4. Section 3.5 presents the experimental methodology (data sets, evaluation methods and performance metrics) used for proof-of-concept computer simulations. Finally, simulation results are presented and discussed in Section 3.6.

3.2 Anomaly Detection with HMMs

A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, and a set of observation probability distributions, each one associated with a state. At each discrete time instant, the process is assumed to be in a state, and an observation is generated by the probability distribution corresponding to the current state. HMM parameters are usually trained using the Baum-Welch (BW) algorithm (Baum et al., 1970) – a specialized expectation maximization technique to estimate the parameters of the model from the training data. Theoretical and empirical results have shown that, given an adequate number of states and a sufficiently rich set of observations, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in terms of simple and compact models, with tolerance to noise and uncertainty. For further details regarding HMM the reader is referred to the extensive literature (Ephraim and Merhav, 2002; Rabiner, 1989).

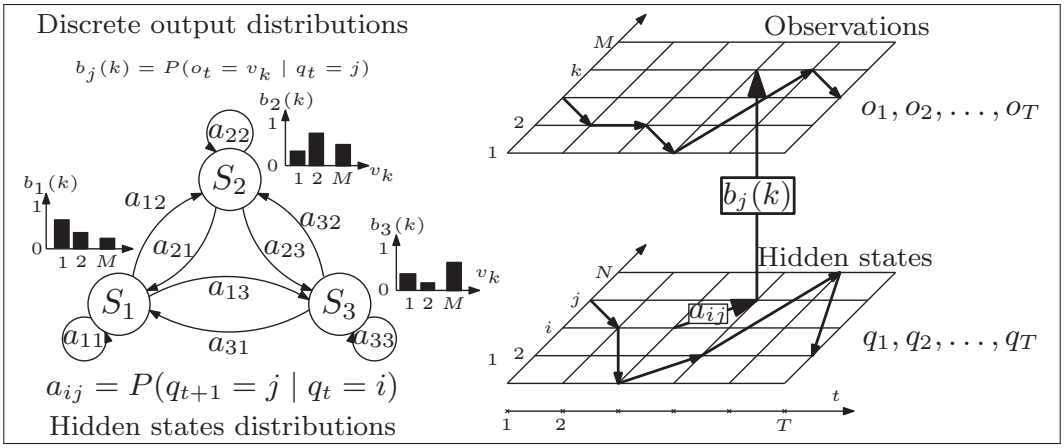


Figure 3.1 Illustration of a fully connected three state HMM with a discrete output observations (left). Illustration of a discrete HMM with N states and M symbols switching between the hidden states q_t and generating the observations o_t (right). The state $q_t = i$ denotes that the state of the process at time t is S_i

Formally, a discrete-time finite-state HMM consists of N hidden states in the finite-state space $S = \{S_1, S_2, ..., S_N\}$ of the Markov process. Starting from an initial state S_i,

determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij} $(1 \leq i, j \leq N)$. As illustrated in Figure 3.1, the process then emits a symbol v according to the output probability distribution $b_j(v)$, which may be discrete or continuous, of the current state S_j. The model is therefore parametrized by the set $\lambda = (\pi, A, B)$, where vector $\pi = \{\pi_i\}$ is initial state probability distribution, matrix $A = \{a_{ij}\}$ denotes the state transition probability distribution, and matrix $B = \{b_j(k)\}$ is the state output probability distribution.

Estimating the parameters of a HMM requires the specification of its order (i.e., the number of hidden states N). The value of N may have a significant impact on both detection performance and training time. The time and memory complexity of BW training is $\mathcal{O}(N^2T)$ and $\mathcal{O}(NT)$ respectively, for a sequence of length T symbols. In the literature on HMMs applied to anomaly detection (Gao et al., 2002; Hoang and Hu, 2004; Warrender et al., 1999), the number of states is often overlooked and typically chosen heuristically. In addition, a single "best" HMM will not provide a high level of performance over the entire detection space. A multiple-HMMs (μ-HMMs) approach, where each HMM is trained using a different number of hidden states, and where HMM responses are combined according to the MRROC has significantly improved performance over a single best HMM and STIDE (Khreich et al., 2009b).

3.3 Fusion of Detectors in the Receiver Operating Characteristic (ROC) Space

A *crisp* detector (e.g., STIDE) outputs only a class label and produces a single operational data point in the ROC plane. In contrast, a *soft* detector (e.g., HMM) assigns scores or probabilities to the input samples, which can be converted to a crisp detector by setting a threshold on the scores. Given a detector and a set of test samples, the true positive rate (tpr) is the proportion of positives correctly classified over the total number of positive samples. The false positive rate (fpr) is the proportion of negatives incorrectly

classified (as positives) over the total number of negative samples. A ROC curve is a two-dimensional curve in which the tpr is plotted against the fpr. A parametric ROC curve typically assumes that a pair of normal distributions underlies the data (Hanley, 1988; Metz, 1978) and increases monotonically with the fpr. In practice, an empirical ROC curve may be obtained by connecting the observed (tpr, fpr) pairs of detectors, therefore it makes minimal assumptions (Fawcett, 2004). Given two operating points, say i and j, in the ROC space, i is defined as *superior* to j if $fpr_i \leq fpr_j$ and $tpr_i \geq tpr_j$. If one ROC curve has all its points superior to those of another curve, it *dominates* the latter. If a ROC curve has $tpr_i > fpr_i$ for all its points i then, it is a *proper* ROC curve. Finally, the area under the ROC curve (AUC) is the fraction of positive–negative pairs that are ranked correctly (see Section 3.5.3).

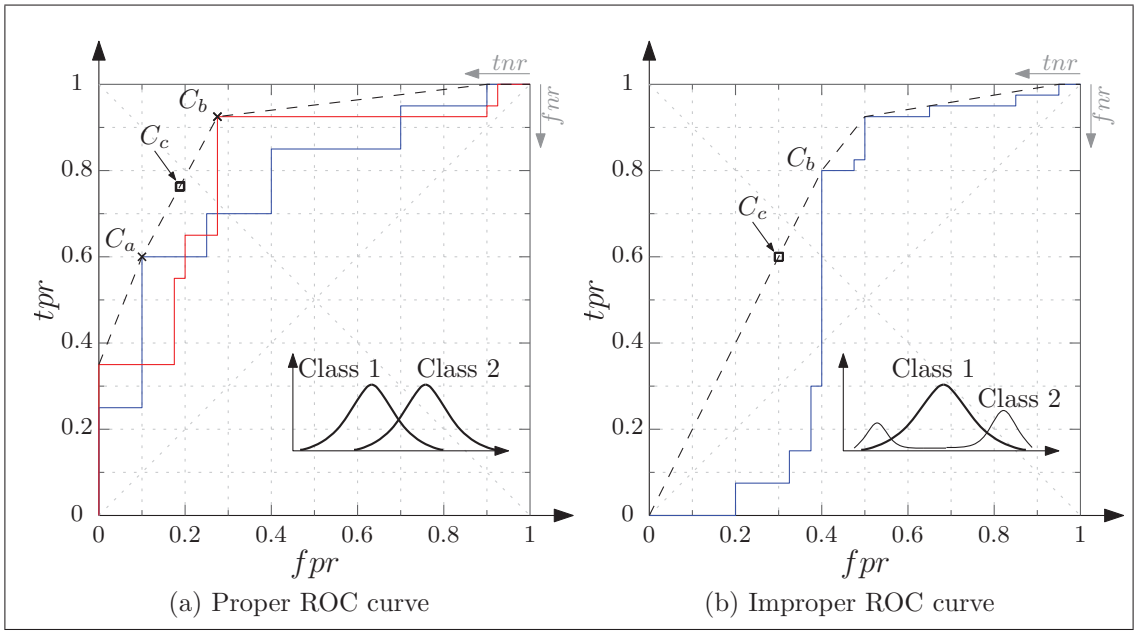
The rest of this section provides an overview of techniques in literature for combining detectors and repairing curves within the ROC space. It includes the stochastic interpolation of the Maximum Realizable ROC (MRROC) or ROCCH, and the conjunction and disjunction rules for combining crisp and soft detectors.

3.3.1 Maximum Realizable ROC (MRROC)

Given that the underlying distributions are generally considered fixed, a parametric ROC curve will always be proper and convex. In practice, an ROC plot is a step-like function which approaches a true curve as the number of samples approaches infinity. An empirical ROC curve is therefore not necessarily convex and proper as illustrated in Figure 3.2. Concavities indicate local performance that is worse than random behavior. These concavities occur with unequal variances between classes, with large skew in the data, or when the classifier is unable to capture the modality of the data (e.g., classifying multimodal data with a linear classifier).

As shown in Figure 3.2, the ROCCH of an empirical ROC is the piece-wise outer envelope connecting only the superior points of an ROC with straight lines. The ROCCH of a single crisp detector connects its resulting point to the $(0,0)$ and $(1,1)$ points. The ROCCH

is also applicable to multiple detectors which may be crisp or soft. If one detector has a dominant ROC curve over fpr values, its convex hull constitutes the overall ROCCH. When there is no clear dominance among multiple detectors across different regions of the ROC space, the ROCCH corresponds to the envelope connecting the superior points in each region (Figure 3.2a). The ROCCH formulation has been proven to be robust in imprecise environments (Provost and Fawcett, 1997). As environmental conditions change, such as prior probabilities and/or costs of errors, only the portion of interest will change, not the hull itself. This change of conditions may lead to shifting the optimal operating point to another threshold or classifier on the convex hull.



(a) Proper ROC curve (b) Improper ROC curve

Figure 3.2 Illustration of the ROCCH (dashed line) applied to: (a) the combination of two soft detectors (b) the repair of concavities of an improper soft detector. For instance, in (a) the composite detector C_c is realized by randomly selecting the responses from C_a half of the time and the other half from C_b

The ROCCH is useful for combining detectors. The idea is based on a simple interpolation between the corresponding detectors (Provost and Fawcett, 2001; Scott et al., 1998). In practice, this is achieved by randomly alternating detectors responses proportionately between the two corresponding vertices of the line segment on the convex hull where the desired operational point lies. This approach has been called the maximum realizable

ROC (MRROC) by Scott et al. (1998) since it represents a system that is equal to, or better than, all the existing systems for all Neyman-Pearson criteria. Hereafter, the acronyms ROCCH and MRROC will be used interchangeably. The performance of the composite detector C_c can be readily set to any point along the line segment, connecting C_a and C_b, simply by varying the desired fpr_c and thus the ratio between the number of times one detector is used relative to the other.

The MRROC considers only the responses of detectors that lie on the facet of the ROCCH, since they are potentially "optimal", and discards the responses not touching the ROCCH (Provost and Fawcett, 1997; Scott et al., 1998). However, this may degrade the performance due to loss of information since inferior detectors are not exploited for decisions. Indeed, these detectors may contain valuable *diverse* information. As detailed next, some combination techniques have been proposed in literature to improve upon the ROCCH.

3.3.2 Repairing Concavities

Repairing concavities is useful in itself for situations with only one detector with some concavities in its ROC curve. Repairing these concavities could be useful to improve the performance. The MRROC may be used to repair the ROC concavities by discretizing and interpolating between thresholds of superior points. Flach and Wu proposed a technique for largest concavity repairing (LCR) (Flach and Wu, 2005). It consists of inverting the largest concavity section with reference to the mid point of the local line segment on the ROCCH. The intuition is that points underneath the ROCCH can be mirrored in the same way negative classifiers (under the ascending diagonal) can be inverted. The LCR consist in determining the two thresholds on the ROCCH, limiting the section to be inverted, and then in negating the responses with reference to the mid point of the corresponding line segment of the ROCCH. In some cases the LCR may provide a higher level of performance than the MRROC.

3.3.3 Conjunction and Disjunction Rules for Crisp Detectors

The Boolean conjunction (AND) and disjunction (OR) fusion functions were first introduced for combining crisp detectors (Daugman, 2000). Other authors such as Fawcett (2004) have noted that it is sometimes possible to find nonlinear combinations of detectors which produce an ROC exceeding their convex hulls.

As illustrated in Figure 3.3, the conjunction rule decreases the fpr at the expenses of decreasing the tpr, thus providing a more conservative performance than each of the original detectors. Analogously, the disjunction rule increases the tpr at the expenses of increasing the fpr, providing a more aggressive performance that each of the original detectors. When these fusions are considered alone – outside the ROC space – their achieved performance may not be of considerable interest as advocated by Daugman (2000). However, depending on detector interaction, within the ROC space, these fusion rules may produce a new convex hull that is superior than that of existing detectors alone. In addition, the new MRROC curve provides the flexibility of choosing any operating point which lies on its hull by interpolating between the relevant vertices as described previously.

The conditional independence assumption among the detectors simplifies the computation. In this cases, the combination rules depend only on the true and false positive rates. Let (tpr_a, fpr_a) and (tpr_b, fpr_b) be the true positive and false positive rates of detectors C_a and C_b, respectively. Under the conditional assumption, the performance of the composite crisp detectors C_c is given in Table 3.1 (Black and Craig, 2002; Fawcett, 2004). These formulas stem from the conditional independence of probability. For instance, the probability that both detectors correctly classify a positive test sample is given by:

$$P_{11|1} = \quad \Pr(C_a = 1, C_b = 1 \mid 1) = \Pr(C_a = 1 \mid 1) \Pr(C_b = 1 \mid 1) = tpr_a tpr_b.$$

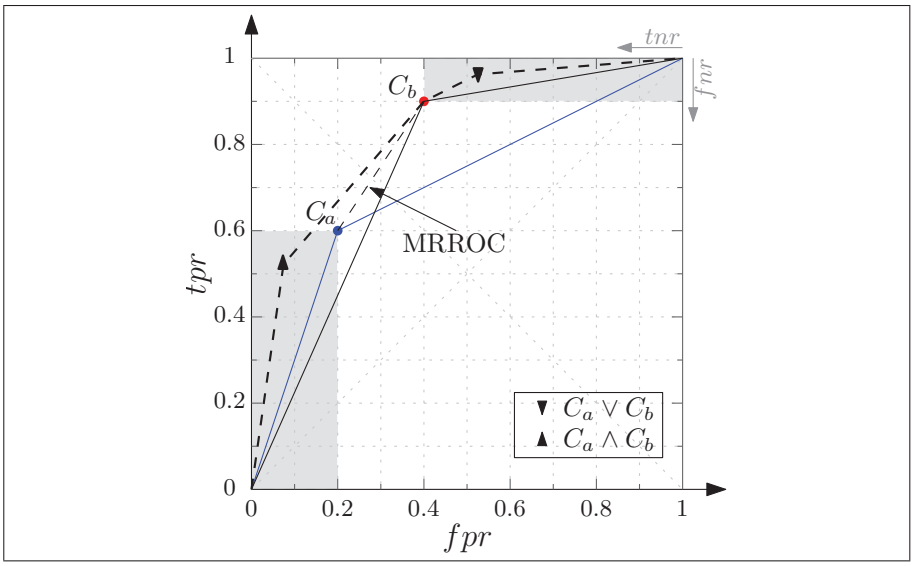


Figure 3.3 Examples of combination of two conditionally-independent crisp detectors, C_a and C_b, using the AND and OR rules. The performance of the their combination is shown superior to that of the MRROC. The shaded regions are the expected performance of combination when there is an interaction between the detectors

Table 3.1 Combination of conditionally independent detectors

$P_{11\|1} = tpr_a tpr_b$	$P_{11\|0} = fpr_a fpr_b$
$P_{10\|1} = (1-tpr_a)tpr_b$	$P_{10\|0} = (1-fpr_a)fpr_b$
$P_{01\|1} = tpr_a(1-tpr_b)$	$P_{01\|0} = fpr_a(1-fpr_b)$
$P_{00\|1} = (1-tpr_a)(1-tpr_a)$	$P_{00\|0} = (1-fpr_a)(1-fpr_b)$

As mentioned, this assumption is violated in most real-world applications. In the more realistic conditionally dependent case, the performance of the composite crisp detectors C_c is given in Table 3.2. The performance now depends on the positive ($P_{11|1}$) and negative ($P_{00|0}$) correlations between detectors (Black and Craig, 2002). That is, the joint distributions of both detectors are required, and the performance can now be anywhere in the shaded regions of Figure 3.3.

An attempt to characterize this dependence is given by Venkataramani and Kumar (2006), where the correlation coefficient between the detector scores is shown to be useful for predicting the "best" decision fusion rule, as well as for evaluating the quality of

Table 3.2 Combination of conditionally dependent detectors

$tpr_a tpr_b < P_{11\|1} < \min(tpr_a, tpr_b)$
$P_{10\|1} = tpr_a - P_{11\|1}$
$P_{01\|1} = tpr_b - P_{11\|1}$
$P_{00\|1} = 1 - tpr_a - tpr_a + P_{11\|1}$
$(1-fpr_a)(1-fpr_b) < P_{00\|0} < \min(1-fpr_a, 1-fpr_b)$
$P_{10\|0} = (1-fpr_b) - P_{00\|0}$
$P_{01\|0} = (1-fpr_a) - P_{00\|0}$
$P_{11\|0} = fpr_a + fpr_b - 1 + P_{00\|0}$

detectors. More recently, in order to avoid the restrictive conditional assumption among detectors, the combination rules were extended to include all Boolean functions (Barreno et al., 2008). By ranking these combinations according to their likelihood ratios, the optimal rules can be obtained. However, due to the doubly exponential explosion of combinations – for n detectors there is 2^n possible outputs resulting in 2^{2^n} possible combinations – the proposed global search for the optimal rules is impractical.

3.3.4 Conjunction and Disjunction Rules for Combining Soft Detectors

Several authors have proposed the application of the Boolean AND and OR fusion functions to combine soft detectors. For a pair-wise combination, the fusion function is applied to each threshold on the first ROC curve with respect to each threshold on of the second curve. The optimum threshold, as well as the combination function, is then found according to the Neyman-Person test (Neyman and Pearson, 1933). That is for each value of the fpr, the point which has the maximum tpr value is selected, along with the corresponding thresholds and Boolean function to be used during operations.

Haker et al. (2005) proposed to apply the AND and OR functions to combine a pair of soft detectors under the assumption of conditional independence between detectors (Table 3.1), and when both detectors are proper and convex. The authors proposed a set of "maximum likelihood combination" rules (see Table 3.3) to select the combination rules or original detectors to be employed. For instance, the AND rule is selected if

Table 3.3 The maximum likelihood combination of detectors C_a and C_b as proposed by Haker et al. (2005)

C_a	C_b	Selection rules for C_c
1	1	$tpr_a tpr_b \geq fpr_a fpr_b$
1	0	$tpr_a(1-tpr_b) \geq fpr_a(1-fpr_b)$
0	1	$(1-tpr_a)tpr_b \geq (1-fpr_a)fpr_b$
0	0	$(1-tpr_a)(1-tpr_b) \geq (1-fpr_a)(1-fpr_b)$

the first condition ($C_a = 1, C_b = 1$ in Table 3.3) is exclusively true, while the OR rule is selected when the first three conditions are true. Otherwise one of the individual detectors is selected. This selection may however discard important combinations since for two thresholds several combination rules may emerge. For example, by computing these theoretical conditions for the ROC curves shown in Figure 3.3, one can observe that the first and third conditions are verified in Table 3.3, hence only C_b is considered for these two points. However, as shown in Figure 3.3, both the AND and OR combination improve the performance over the original detectors.

Tao and Veldhuis (2008) applied and compared AND versus OR Boolean function separately for combining multiple ROC curves using a pair-wise combination. They showed that the OR rule emerges most of the time in their biometrics application. Oxley et al. (2007) proved that, under the independence assumption between detectors, a Boolean algebra of families of classification systems is isomorphic to a Boolean algebra of ROC curves. Shen (2008) tried to characterize the effect of correlation on the AND and OR combination rules, using a bivariate normal model. He showed that discrimination power is higher when the correlation is of opposite sign.

Most research has addressed the problem of Boolean combinations under the assumption of smooth, convex and proper ROC curves. Such curves results from a parametric models, or when the data is abundant for both classes. In the ideal case, when both conditional independence and convexity assumptions are fulfilled, the AND and OR combinations are proven to be optimal, providing a higher level of performance than the original ROC curves (Barreno et al., 2008; Thomopoulos et al., 1989; Varshney, 1997). When provided

with limited and imbalanced data for training and validation, the ROC curves may be improper and large concavities will appear. When either one of the assumptions is violated, the performance of these combinations will be sub-optimal. In contrast to crisp detectors, the correlation between soft detector decisions also depends on the thresholds selection. At different thresholds on two ROC curves the conditional independence may be violated and different type of correlations may be introduced.

3.4 A Boolean Combination (BC_{ALL}) Algorithm for Fusion of Detectors

3.4.1 Boolean Combination of Two ROC Curves

In this chapter, a general technique for Boolean combination (BC_{ALL}) is proposed for fusion of detector responses in the ROC space. In particular, this algorithm can exploit information from the ROC curves when detectors are trained from limited and imbalanced data. In this case, the ROC curves typically comprise concavities and are not necessarily proper. Figure 3.4 presents the block diagram of a system that combines the responses of two HMMs in the ROC space according to the BC_{ALL} technique. It involves fusing responses of detectors using all Boolean functions, prior to applying the MRROC. In contrast with most work in literature, where either AND or OR functions are applied, the BC_{ALL} technique takes advantage of all Boolean functions applied to the ROC curves and selects those that improve the ROCCH.

The main steps of BC_{ALL} are presented in Algorithm 3.1. The BC_{ALL} technique inputs a pair of ROC curves defined by their decision thresholds, T_a and T_b, and the labels for the validation set. Using each of the ten Boolean functions (refer to IV.1), BC_{ALL} fuses the responses of each threshold from the first curve (R_{a_i}) with the responses of each threshold from the second (R_{b_i})[4]. Responses of the fused thresholds are then mapped to points (fpr, tpr) in the ROC space. The thresholds of points that exceeded the original ROCCH

[4]For each ROC curve, the matrix of responses associated with the thresholds can be directly input to Algorithm 3.1 instead of converting each threshold to corresponding responses at line 8 and 10. Although it is not useful for combining two curves and it increases the memory requirements, the matrix of responses is needed to recombine resulting responses with another curve as in the following algorithms.

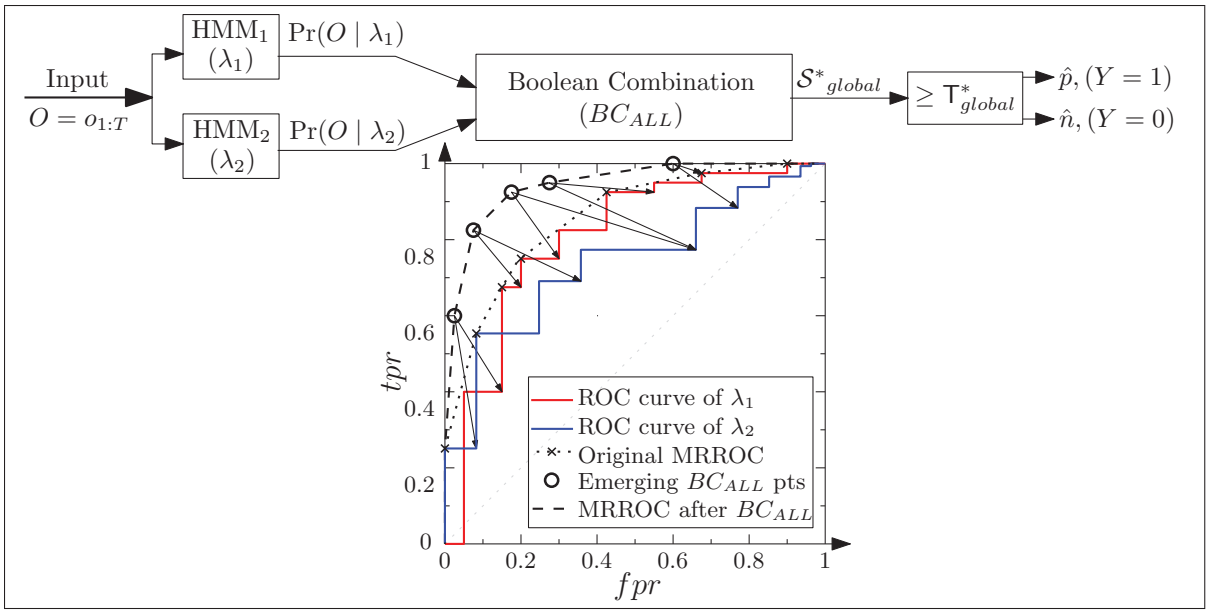


Figure 3.4 Block diagram of the system used for combining the responses of two HMMs. It illustrates the combination of HMM responses in the ROC space according to the BC_{ALL} technique

of original curves are then stored along with their corresponding Boolean functions. The ROCCH is then updated to include the new emerging points. When the algorithm stops, the final ROCCH is the new MRROC in the Newman-Pearson sense. The outputs are the vertices of the final ROCCH, where each point is the results of two thresholds from the ROC curves fused with the corresponding Boolean function. These thresholds and Boolean functions form the elements of $\mathcal{S}^*_{global}$, and are stored and applied during operations, as illustrated in Figure 3.4.

The BC_{ALL} technique makes no assumptions regarding the independence of the detectors. Instead of fusing the points of ROC curves under the independence assumptions (Table 3.1), this techniques directly fuses the responses of each decision threshold, accounting for both independent and dependent cases. In fact, by applying all Boolean functions to combine the responses for each threshold (line 11 of Algorithm 3.1), it implicitly accounts for the effects of correlation (see Table 3.2). This is due to the direct fusion of responses, which considers the joint conditional probabilities of each detector at each threshold. Furthermore, the BC_{ALL} always provides a level of performance that

Algorithm 3.1: $BC_{ALL}(\mathsf{T}_a, \mathsf{T}_b, labels)$: Boolean combination of two ROC curves

Input: Thresholds of ROC curves, T_a and T_b, and $labels$ (of validation set)
Output: ROCCH and fused responses (Rab) of combined curves, where each point is the result of two fused thresholds along with the corresponding Boolean function (bf)

let $m \leftarrow$ number of distinct thresholds in T_a
let $n \leftarrow$ number of distinct thresholds in T_b
Allocate F an array of size: $[2, m \times n]$ `// holds temporary results of fusions`
$BooleanFunctions \leftarrow \{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$
Compute $ROCCH_{old}$ of the original curves
foreach $bf \in BooleanFunctions$ **do**
    for $i = 1, \ldots, m$ **do**
        $R_a \leftarrow (\mathsf{T}_a \geq \mathsf{T}_{a_i})$ `// converting threshold of 1st ROC to responses`
        for $j = 1, \ldots, n$ **do**
            $R_b \leftarrow (\mathsf{T}_b \geq \mathsf{T}_{b_j})$ `// converting threshold of 2nd ROC to responses`
            $R_c \leftarrow bf(R_a, R_b)$ `// combined responses with` bf
            Compute (tpr, fpr) using R_c and $labels$
            Push (tpr, fpr) onto F
    Compute $ROCCH_{new}$ of F
    Store thresholds and corresponding Boolean functions that exeeded the $ROCCH_{old}$, $\mathcal{S}^*_{global} \leftarrow (\mathsf{T}_{a_x}, \mathsf{T}_{b_y}, bf)$ `// to be used during operations`
    Store the responses of these emerging points into R `// to be used with` BCM_{ALL} `and` IBC_{ALL}
    $ROCCH_{new} \leftarrow ROCCH_{old}$ `// Update ROCCH`
Return $ROCCH_{new}, R, \mathcal{S}^*_{global}$

is equivalent or higher than that of the MRROC of the original ROC curves. In the worst-case scenario, when the responses of detectors do not provide diverse information, or when the shape of the ROC curve on the validation set differs significantly from that of the test set, the BC_{ALL} is lower bounded by the MRROC of the original curves.

Including all Boolean functions accommodates for the concavities in the curves. Indeed, AND and OR rules will not provide improvements for the inferior points that correspond to concavities and make for an improper ROC curve, or points that are close to the diagonal line in the ROC space. Other Boolean functions, for instance those that exploit negations of responses, may however emerge. The BC_{ALL} technique can therefore be applied even when training and validation data are limited and heavily imbalanced,

to combine the decisions of any soft, crisp, or hybrid detectors in the ROC space. This includes combining the responses of the same detector trained on different data or features or trained according to different parameters, or from different detectors trained on the same data, etc.

3.4.2 Boolean Combination of Multiple ROC Curves

Different strategies may be implemented for combining multi-ROC curves. A commonly proposed strategy for a cumulative combination of ROC curves is to start by any pair of the ROC curves then combine the resulting responses with the third, then with the fourth and so on, until the last ROC curve (Pepe and Thompson, 2000; Tao and Veldhuis, 2008). As described in Algorithm 3.2, the thresholds (T_1 and T_2) of first two ROC curves are initially combined with the BC_{ALL} technique. Then, their combined responses (R_1) are directly input into line 8 of Algorithm 3.1 and combined with the thresholds of the third ROC curve (T_3). A pair-wise combination of ROC curves is another alternative, in which the BC_{ALL} technique is applied to each pair of ROC curves, and then the MRROC is then applied to the resulting combinations. Both strategies have been investigated and typically lead to comparable results. However, the time and memory complexity associated with the cumulative strategy can be considerably lower than for the pair-wise one. This is due to the number of permutations required in the pair-wise combinations. In additions, the pair-wise strategy requires combining all thresholds for each two curves, while combining the resulting responses with a new curve is less demanding since the number of selected responses is typically much lower than the number of thresholds. The reader is referred to Subsection 3.4.3 for additional details. The cumulative strategy for Boolean Combination of Multiple ROC curves (BCM_{ALL}) described in Algorithm 3.2 is adopted in this chapter.

Further improvements in performance may be achieved by re-combining the output responses of combinations resulting from the BC_{ALL} (or BCM_{ALL}) with those of the original ROC curves over several iterations. A novel Iterative Boolean Combination

Algorithm 3.2: $BCM_{ALL}(\mathsf{T}_1,\ldots,\mathsf{T}_K,labels)$: Cummulative combination of multiple ROC curves based on BC_{ALL}

Input: Thresholds of K ROC curves $[\mathsf{T}_1,\ldots,\mathsf{T}_K]$ and *labels*
Output: ROCCH of combined curves where each point is the result of the combination of combinations

1 $[ROCCH_1, R_1] = BC_{ALL}(\mathsf{T}_1,\mathsf{T}_2,labels)$ `// combine the first two ROC curves`
2 **for** $k = 3,\ldots,K$ **do**
    `// combine the responses of the previous combination with those of the following ROC curve`
3    $[ROCCH_{k-1}, R_{k-1}] = BC_{ALL}(R_{k-2},\mathsf{T}_k,labels)$
4 **Return** $ROCCH_{K-1}, R_{K-1}$ and the stored tree of the selected responses/thresholds fusions along with their corresponding fusion functions

(IBC_{ALL}) is presented in Algorithm 3.3 and allows for combination that maximize the AUC of K ROC curves by re-combining the previously selected thresholds and fusion functions with those of the original ROC curves. During the first iteration, the ROC curves of two or more detectors are combined using the BC_{ALL} or BCM_{ALL}. This defines a potential direction for further improvement in performance within the combination space. Then, the IBC_{ALL} proceeds in this direction by re-considering information from the original curves over several iterations. The iterative procedure accounts for potential combinations that may have been disregarded during the first iteration, and are mostly useful when provided with limited and imbalanced training data. The iterative procedure stops when there are no further improvements between the AUC of old and new ROCCHs or a maximum number of iterations are performed. This stopping criteria can be controlled by tolerating a small difference between the old and new AUC values ($\epsilon = AUC(ROCCH_{NEW}) - AUC(ROCCH_{OLD})$), or by applying a statistical test for significance when working with several replications. Although sub-optimal, the IBC_{ALL} algorithm overcomes the impractical exponential explosion in computational complexity associated with the brute-force strategy suggested by Barreno et al. (2008) (see Subsection 3.4.3).

Note that the IBC_{ALL} can also be applied to repair ROC concavities. In such scenarios, the same thresholds, say T_a, of the ROC curve to be repaired are input twice into the IBC_{ALL} algorithm, i.e., $IBC_{ALL}(\mathsf{T}_a,\mathsf{T}_a,labels)$, and iterates until the AUC stops

Algorithm 3.3: $IBC_{ALL}(\mathsf{T}_1,\ldots,\mathsf{T}_K,labels)$: Iterative Boolean combination based on BC_{ALL} or BC_{ALL}

Input: Thresholds of K ROC curves $[\mathsf{T}_1,\ldots,\mathsf{T}_K]$ and $labels$
Output: ROCCH of combined curves where each point is the result of the combination of combinations through several iterations

1 $[ROCCH_{OLD}, R_{OLD}] = BCM([\mathsf{T}_1,\mathsf{T}_2,\ldots,\mathsf{T}_K],labels)$
2 **while** $(AUC(ROCCH_{NEW}) \geq AUC(ROCCH_{OLD}) + \epsilon)$ *or* $(numberIterations \leq maxIter)$ **do**
3 $\quad [ROCCH_{NEW}, R_{NEW}] = BC(R_{OLD},[\mathsf{T}_1,\mathsf{T}_2,\ldots,\mathsf{T}_K],labels)$
4 **return** $ROCCH_{NEW}, R_{NEW}$ and the stored tree of the selected responses fusions along with their corresponding fusion functions

improving. After applying the IBC_{ALL} to a ROC curve the diverse information from the inferior points are taken into consideration in view improving the performance. The resulting MRROC curve is guaranteed to be proper and convex. In the worst-case scenario, the repaired is lower bounded by the ROCCH. Finally, the IBC_{ALL} repairs the concavities in a complementary way to LCR (Flach and Wu, 2005), therefore applying both techniques may yield even higher level of performance as presented in Section 3.6.

3.4.3 Time and Memory Complexity

Given a pair of detectors, C_a and C_b, having respectively n_a and n_b distinct thresholds on their ROC curves. During the design of the IBC_{ALL} system, the worst-case time required for fusion using BC_{ALL} is the time required for computing all ten Boolean functions to combine those thresholds, i.e., $10 \times n_b \times n_b$. The worst-case time complexity is $\mathcal{O}(n_a n_b)$ Boolean operations. The worst-case memory requirements is an array of floating point registers of size $2 \times n_b \times n_b$ for storing the temporary results (tpr, fpr) of each Boolean function (denoted by F in Algorithm 3.1). Therefore, the worst-case memory complexity is $\mathcal{O}(n_a n_b)$.

When the number of distinct thresholds becomes very large, these thresholds can be sampled (or histogrammed) into a smaller number of bins before applying the algorithm to reduce both time and memory complexity. Nevertheless, in scenarios with limited and

imbalanced data, which is the main focus of this work, the number of distinct thresholds is typically small. The BC_{ALL} is very efficient in these cases.

When the BCM_{ALL} is applied to combine the response of several ROC curves of K detectors, the worst-case time can be roughly stated as K times that of the BC_{ALL} algorithm. However, after combining the first two ROC curves, the number of emerging responses on the ROCCH, is typically very small with respect to the number of thresholds on each ROC curve. Let n_{max} be the largest number of thresholds among the K ROC curves to be combined. When K grows, it is conservative to consider the worst-case time complexity of the order of $\mathcal{O}(n_{max}^2 + K.n_{max})$ Boolean operations. The worst-case time complexity for combining K detectors with IBC_{ALL} is that of the BCM_{ALL} multiplied with the number of iterations (I), $\mathcal{O}(I(n_{max}^2 + K.n_{max}))$. The worst-case memory complexity for both BCM_{ALL} and IBC_{ALL} is $\mathcal{O}(n_{max}n_{max})$. This is limited to the memory required for combining the first two ROC curves with the BC_{ALL} algorithm.

As a comparative example, consider two soft detectors C_a and C_b with their ROC curves having respectively a *small* number of distinct thresholds $n_a = 100$ and $n_b = 50$. Since each threshold on the ROC curve of a soft detector is a crisp detector, the total number of crisp detectors is therefore $n = n_a + n_b = 150$. The brute-force search for optimal combination of crisp detector is 2^{2^n} and can be reduced to 2^n as proposed by Barreno et al. (2008). The exhaustive optimal search requires a prohibitively large number, $2^{150} \approx 1.4 \times 10^{45}$, of Boolean computationsauthors stated that for only $n = 40$ detectors the computational time would require about a year and a half (Barreno et al., 2008).$5,000$ Boolean computations with BC_{ALL} and $I \times 10,200 \approx 10^6$ Boolean computations with IBC_{ALL}, where the number of iterations I is typically less than ten. Although the IBC_{ALL} algorithm is efficient, its time and memory complexity can be always reduced using the sampling technique, on the account of a small loss in the combination performance.

In contrast, during operations the system will be using one vertex or interpolating between two vertices on the final convex hull provided by IBC_{ALL} according to a specific

false alarm rate. Each vertex has its own set of Boolean combination functions, which may be derived from all (or a subset of) K detectors that have been considered during the design phase. In practice, the computational overhead of these Boolean functions is lower than that of operating the required number of detectors. Therefore, the worst-case time and memory complexity is limited to operating the K detectors. When there are design constraints on the number of operational detectors K, they must be considered during the design phase. A larger set of detectors $\mathcal{K} > K$ could be first employed to realize an upper bound for analyzing the system performance. Then, the best subset of K detectors that limits the decrease of performance with reference to the upper bound can be selected for operations. This trade-off between the number of operational detectors and required performance can be a time consuming task. However since it is conducted during the design phase, various subsets selection and parallel processing techniques can be employed for a more efficient search.

3.4.4 Related Work on Classifiers Combinations

Classifiers can be combined at various levels and categorized according to pre- and post-classification levels (Kuncheva, 2004a; Tulyakov et al., 2008). In pre-classification combination occurs at the sensor (or raw data) and feature levels, while post-classification combination occurs at the score, rank and decision levels. Pre-classification fusions methods are based on the generation of ensemble of classifiers (EoC), each trained on different data sets or subsets obtained by using techniques such as data-splitting, cross-validation, bagging (Breiman, 1996), and boosting (Freund and Schapire, 1996). This can be also obtained by constructing classifiers that are trained on different features subsets, for instance ensemble generation methods such as the random subspace method (Ho, 1998). Static ensemble selection attempts to select the "best" classifiers from the pool based on various diversity measures (Brown et al., 2005; Kuncheva and Whitaker, 2003), prior to combining their results. An alternative approach consists in combining the outputs of classifiers and then selecting the best performing ensemble evaluated on an independent validation set (Banfield et al., 2003; Ruta and Gabrys, 2005). The later approach has

proven to be more reliable than the diversity-based approach (Kuncheva and Whitaker, 2003; Ruta and Gabrys, 2005). However, since combination is performed before selection its success depends on the chosen method(s) of combination, which may be sub-optimal.

In post-classification phase, whether the EoC is inherent to the problem at hand, generated or selected, classifier responses must be combined using a fusion function. Fusion at the score level is more prevalent in literature (Kittler, 1998). Normalization of the scores is typically required, which may not be a trivial task. Fusion functions may be static (e.g., sum, product, min, max, average, majority vote, etc.), adaptive (e.g., weighted average, weighted vote, etc.) or trainable (also known as stacked or meta-classifier), where another classifier is trained on classifier responses and then used as combiner (Roli et al., 2002; Wolpert, 1992). This trainable approach may introduce overfitting and requires an independent validation set for tuning the combiner parameters. Fusion at the rank level is mostly suitable for multi-class classification problems, where the correct class is expected to appear in the top of the ranked list. Logistic regression and Borda count (Ho et al., 1994; Van Erp and Schomaker, 2000) are among the fusion methods at this level. Rank-level methods simplify the combiner design since normalization is not required.

Fusion at the decision level exploits the least amount of information since only class labels are input to the combiner. Compared to the other fusion methods, it is less investigated in literature. The simple majority voting rule (Ruta and Gabrys, 2002) and behavior-knowledge-space (BKS) (Raudys and Roli, 2003) are the two most representative decision-level fusion methods. One issue that appears with decision level fusion is the possibility of ties. The number of classifiers must therefore be greater than the number of classes. BKS can be only applied to low dimensional problems. Moreover, in order to have an accurate probability estimation, it requires a large number of training samples and another independent database to design the combination rules.

The proposed IBC in this chapter provides an efficient technique which exploit all Boolean combinations as well as the MRROC interpolation for an improved performance.

Combination of responses within the ROC space does not require neither re-training of dichotomizers nor normalization of scores. This is because ROC curves are invariant to monotonic transformation of classification thresholds (Fawcett, 2004). These advantages allow the IBC technique to be directly applied at either the score or the decision levels.

Bayesian learning approaches have been proposed to estimate HMM parameters by integrating over the parameters rather than optimizing. For instance, variational Bayesian learning has been proposed with a suitable prior for all variables to estimate an ensemble of HMMs with the same order, each trained on a different subset of the data, to approximate the entire posterior probability distribution (MacKay, 1997). Nonparametric Bayesian learning have also been proposed for estimating HMMs order and other parameters from the training data (Beal et al., 2002). Starting with a large order, the HMM parameters and the number of states are integrated out with reference to their posterior probabilities. As the method converges to a solution, redundant states are eliminated which yields to automatic order selection. These can been considered as pre-classification combinations of HMMs. HMMs with the same orders are trained and combined using different subsets of the data or HMMs with the different orders are trained and combined on the same data. Comparing the performance of these techniques to that of the IBC would be an interesting future work. Note however that the proposed IBC is a general post-classification combination technique at the response level. It can be used to combine the results of these Bayesian approaches, as well as the results of any other technique, for improved performance.

3.5 Experimental Methodology

The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets[5].

[5] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

3.5.1 University of New Mexico (UNM) Data

The UNM data sets are commonly used for benchmarking anomaly detections based on system calls sequences (Warrender et al., 1999). In related work, intrusive sequences are usually labeled by comparing normal sequences, using STIDE matching technique. This labeling process considers STIDE responses as the ground truth, and leads to a biased evaluation and comparison of techniques, which depends on both training data size and detector window size. To confirm the results on system calls data from real processes, the same labeling strategy is used in this work. However fewer sequences are used to train the HMMs and STIDE to alleviate the bias. Therefore, STIDE is first trained on all the available normal data according to different window sizes, and then used to label the corresponding sub-sequences from the ten sequences available for testing. The resulting labeled sub-sequences of the same size are concatenated, then divided into blocks of equal sizes, one for validation and the other for testing. During the experiments, smaller blocks of normal data (100 to $1,000$ symbols) are used for training the HMMs and STIDE. In addition to the labeling issue, the normal sendmail data is very redundant and anomalous sub-sequences in the testing data are very limited. Nevertheless, due to the limited publicly available system call data, sendmail data is the mostly used in literature.

3.5.2 Synthetic Data

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training and labeling) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations. The data generator is based on the Conditional Relative Entropy (CRE) of a source; it is closely related to the work of Tan and Maxion (Tan and Maxion, 2003). The CRE is defined as the conditional entropy divided by the maximum entropy ($MaxEnt$) of that source, which gives an irregularity index to the generated data.

For two random variables x and y the CRE is given by $CRE = \frac{-\sum_x p(x) \sum_y p(y|x) \log p(y|x)}{MaxEnt}$, where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a sequence of system calls, the conditional probability, $p(y \mid x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $MM = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly sequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly sequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly sequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test[6].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE = 1$) is generated, using the same alphabet size Σ, and segmented into sub-sequences of a desired length using a sliding window with a fixed size of AS. Then, the original generative Markov chain is used to compute the likelihood of each sub-sequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol tran-

[6]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occurs in high frequency over a short period of time.

sitions in the sequence of size AS. This ensures that the anomalous sequences of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into this sequence at random according to a mixing ratio.

Figure 3.5 illustrates the data pre-processing for training, validating and testing using the UNM sendmail data or the generated data. The only difference is the ground truth for labeling, which is all the available normal data for sendmail and the generator itself for the synthetic data.

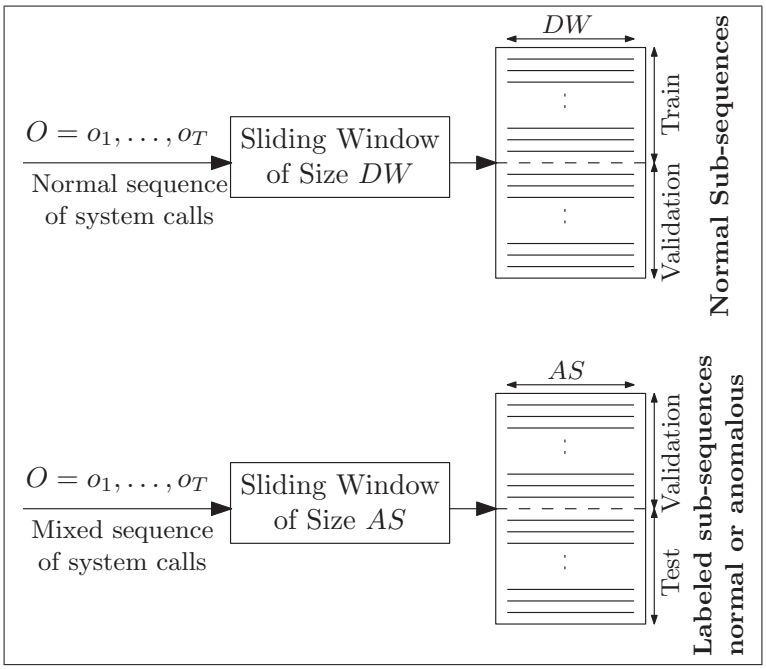


Figure 3.5 Illustration of data pre-processing for training, validation and testing

The experiments conducted in this chapter using the data generator simulate a small process and a more complex process, with $\Sigma = \{8, 50\}$ symbols, and $CRE = \{0.3, 0.4\}$, respectively. The sizes of injected anomalies are assumed equal to the detector window sizes $AS = DW = \{2, 4, 6\}$. For both scenarios, the presented results are for validation and test sets that comprise 75% of normal and 25% of anomalous data.

3.5.3 Experimental Protocol

Figure 3.6 illustrates the steps involved for estimating HMM parameters. For each detector window set of size DW, different discrete-time ergodic HMMs are trained with various number of hidden states $N = [N_{min}, \ldots, N_{max}]$. The number of symbols is taken equal to the process alphabet size. The iterative Baum-Welch algorithm is used to estimate HMM parameters (Baum et al., 1970) using the training data, which only comprises normal sequences. To reduce overfitting effects, the evaluation of the log-likelihood, using the Forward algorithm (Baum et al., 1970), on an independent validation set also comprising only normal sequences is used as a stopping criterion. The training process is repeated ten times using a different random initialization to avoid local minima. The log-likelihood of a second validation set comprising normal and anomalous sequences is then evaluated by each HMM, which provides ten ROC curves. Finally, the model that gives the highest area under its convex hull is selected, which results an HMM for each N value. When working with the synthetic data, this procedure is replicated ten times with different training, validation and testing sets, and the results are averaged and presented along with the standard deviations to provide a statistical confidence intervals.

The performance obtained by fusion of μ-HMMs in ROC space with the proposed BC_{ALL} technique is compared to that of MRROC fusion of the original models, and to that of the conjunction (BC_{AND}) and disjunction (BC_{OR}) combinations (Haker et al., 2005; Tao and Veldhuis, 2008). This is also compared to the performance of the IBC_{ALL} technique applied to combine the μ-HMMs through several iterations. In addition, the performance of STIDE is shown as reference. To investigate the effect of repairing the concavities in

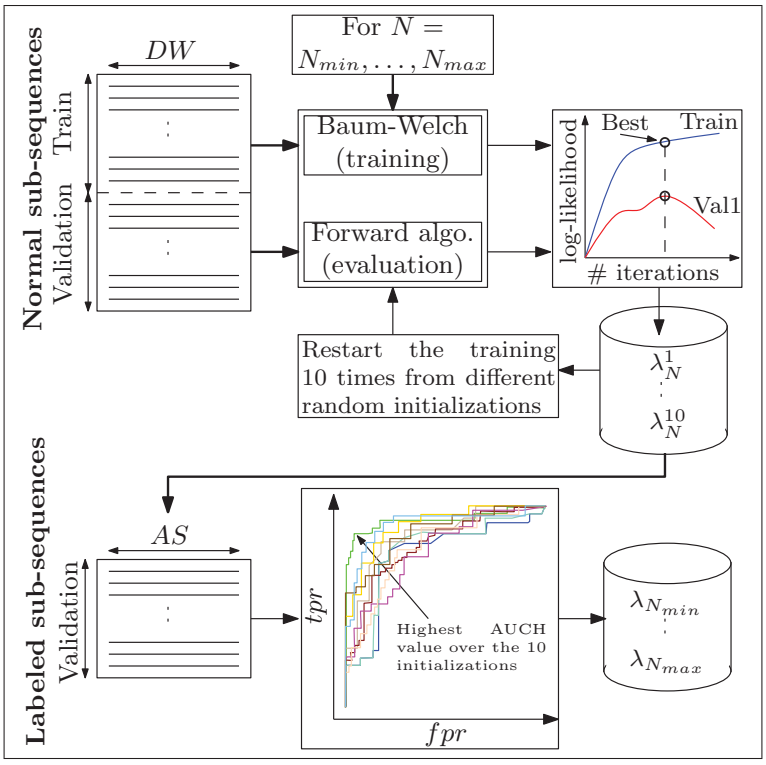


Figure 3.6 Illustration of the steps involved for estimating HMM parameters

ROC curves, each ROC curve is first repaired using the IBC_{ALL} and LCR (Flach and Wu, 2005) techniques, and then all curves are combined with the MRROC technique.

The area under the ROC curve (AUC), has been proposed as more robust scalar summary of classifiers performance than accuracy (Bradley, 1997; Huang and Ling, 2005; Provost and Fawcett, 2001). The AUC assesses the ranking in terms of class separation, i.e., evaluates how well a classifier is able to sort its predictions according to the confidence it assigns to them. For instance, with an $AUC = 1$ all positives are ranked higher than negatives indicating a perfect discrimination between classes. A random classifier has an $AUC = 0.5$ that is both classes are ranked at random. For a crisp classifier, the AUC is simply the area under the trapezoid and is calculated as the average of the tpr and fpr values. For a soft classifier, the AUC may be estimated directly from the data either by summing up the areas of the underlying trapezoids (Fawcett, 2004) or by means of the Wilcoxon–Mann–Whitney (WMW) statistic (Hanley and McNeil, 1982). When the ROC curves cross, It is possible for a high-AUC classifier to perform *worse* in a specific

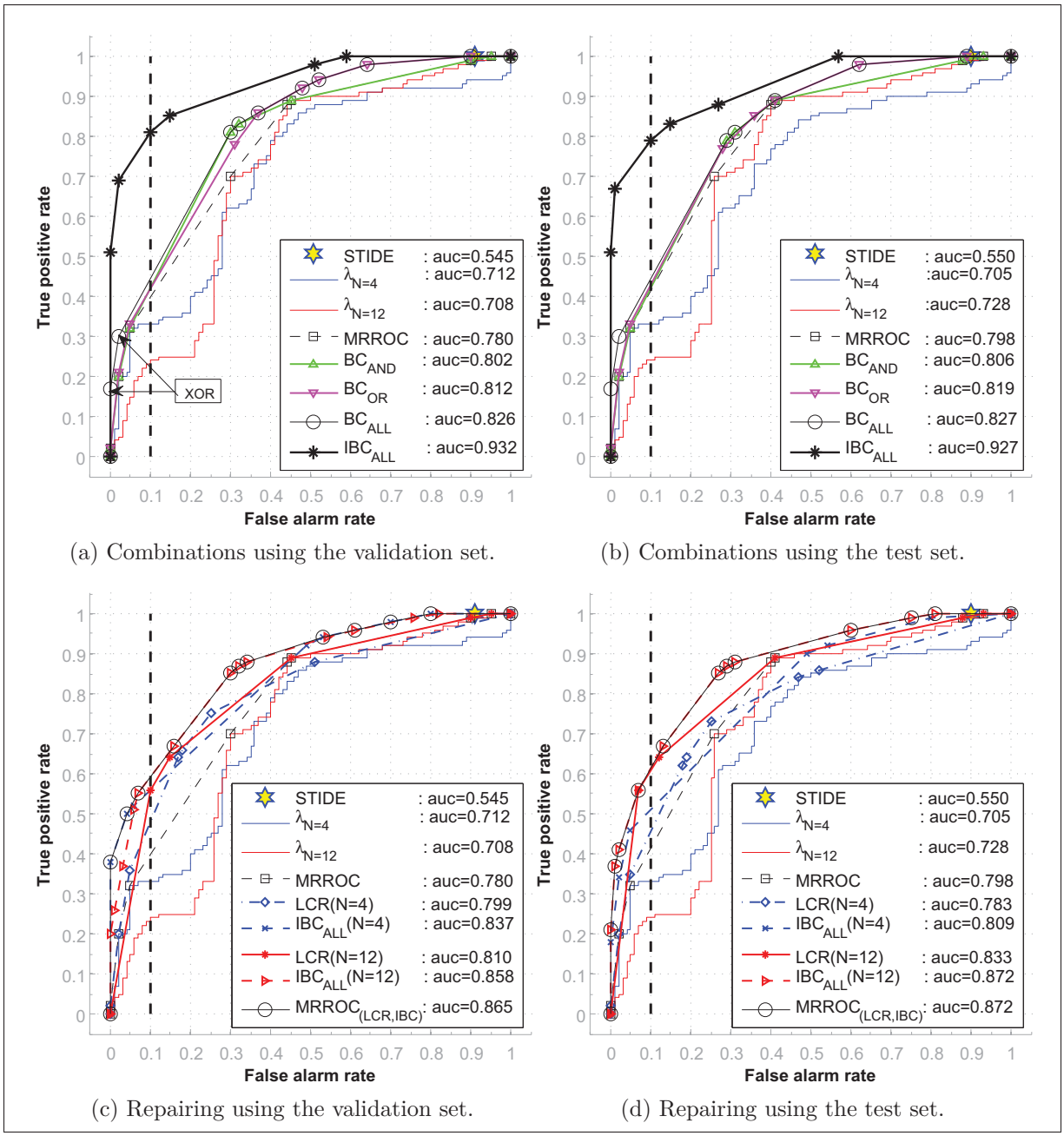
region of ROC space than a low-AUC classifier. In such case, the partial area under the ROC curve (pAUC) (Walter, 2005) could be useful for comparing the specific regions of interest (Zhang et al., 2002). If the AUCs (or the pAUCs) are not significantly different, the shape of the curves might need to be looked at. It may also be useful to look at the *tpr* for a fixed fpr of particular interest. Since the MRROC can be applied to any ROC curve, the performance measures in all experiments are taken with reference to the ROCCH. This includes the area under the convex hull (AUCH), the partial area under the convex hull for the range of $fpr = [0, 0.1]$ ($AUCH_{0.1}$), and the *tpr* at a fixed $fpr = 0.1$.

3.6 Simulation Results and Discussion

3.6.1 An Illustrative Example with Synthetic Data

Figure 3.7 presents an example of the impact on performance obtained after applying different techniques for combining and repairing ROC curves: MRROC, BC and IBC. The training, validation and testing data are generated synthetically as described in Section 3.5.2, with an alphabet of size $\Sigma = 8$ symbols and with a $CRE = 0.3$. The training and validation of the ergodic HMMs is carried out according to the methodology described in Section 3.5. A block of data of size 100 sequences, each of size $DW = 4$, is used to train HMMs, and another validation block of the same size is used to implement a stopping criterion. Each validation and test set is comprised of 200 sequences, each of size $AS = 4$. In both cases, the ratio of normal to anomalous sequences is four to one.

For improved visibility, Figure 3.7 only shows the combinations of two HMMs, each one trained with different number of states, $N = 4$ and 12. The ROC curves for the two HMMs along with their MRROC are presented for the validation (Figure 3.7a) and test (Figure 3.7b) data sets. The performance of STIDE is also shown for reference. To visualize the impact on performance of combining with AND and OR Boolean functions separately (Haker et al., 2005; Tao and Veldhuis, 2008), Figures 3.7a and b show the results with BC_{AND} and BC_{OR} along with BC_{ALL} and IBC_{ALL}. In Figures 3.7c and



(a) Combinations using the validation set.

(b) Combinations using the test set.

(c) Repairing using the validation set.

(d) Repairing using the test set.

Figure 3.7 Illustrative example that compares the AUC performance of techniques for combination (top) and for repairing (bottom) of ROC curves. The example is conducted on a system consisting of two ergodic HMMs trained with $N = 4$ and $N = 12$ states, on a block of 100 sequences, each of length $DW = 4$ symbols, synthetically generated with $\Sigma = 8$ and $CRE = 0.3$

d, IBC_{ALL}, and LCR (Flach and Wu, 2005) are applied to repair each ROC curve, and the resulting curves are then combined with the MRROC technique.

As expected, STIDE performs poorly since the data provided for training is limited and the database memorized by STIDE only comprises a fraction of the normal behavior. This contrasts with the generalization capabilities of HMMs. As shown in Figures 3.7a or b, the improvement in AUC performance achieved by applying BC_{AND} and BC_{OR} is modest. This is also seen when comparing their *tpr* values at an operating point of, for instance, $fpr = 0.1$ with respect to the MRROC of the original ROC curves. Indeed, these AND and OR Boolean functions are unable to exploit information in inferior points. In this case, the BC_{ALL} does not provide a considerably higher level of performance than the BC_{AND} and BC_{OR}, this is not typically the case, but was deliberately selected for this example. However, as shown in Figure 3.7a, two additional combination rules have emerged from the XOR Boolean function with BC_{ALL}. Diverse information from the emerging combination rules (resulting, in this case, from AND, OR and XOR Boolean functions) serve as a guide, within the combination space, to a further performance improvement with the IBC_{ALL} technique. Indeed, after seven iterations, IBC_{ALL} is able to achieve a considerably higher level of performance by recombining emerging rules from each iteration with the original curves. IBC_{ALL} iterative procedure repeatedly selects diverse information and accounts for potential combinations that may have been disregarded during previous iterations.

In this example, the validation set is comprised of 200 sequences and the ROC curves of both HMMs ($\lambda_{N=4}$ and $\lambda_{N=12}$) contain the same number of unique thresholds[7], 200. The worst-case time complexity involved with BC_{ALL} is the time required to compute all ten Boolean functions for each combination of thresholds, i.e., $10 \times 200 \times 200 = 400,000$ Boolean operations. The worst-case memory complexity is an array of floating point registers of size $2 \times 200 \times 200 = 80,000$, holding the temporary results (tpr, fpr) of each Boolean function. This consists the first iteration of IBC_{ALL} and results in nine emerging combinations (or points on the ROCCH) as shown in Figure 3.7a. The time complexity of the second iteration is reduced to the time required for computing $10 \times 9 \times (200 + 200) =$

[7] In many cases, the number of unique thresholds could be lower than the number of samples in the validation set.

Table 3.4 Worst-case time and memory complexity for the illustrative example

Number of iterations	1	2	3	4	5	6	7
Number of emerging points	9	10	10	11	10	9	6
Time complexity	400,000	36,000	40,000	40,000	44,000	40,000	36,000
Memory complexity	80,000	3,600	4,000	4,000	4,400	4,000	3,600

$36,000$ Boolean operations, and the memory complexity is reduced to $2 \times 1,800 = 3,600$ floating point registers. As shown in Table 3.4, the time and memory complexity of successive iterations are reduced by order of magnitude compared to the first iteration. However, as shown in Figure 3.7a, the level of performance is significantly improved due to these low cost iterations. During operations, the number of Boolean combinations varies with the selected operational point, however it is upper-bounded by the number of iterations. For instance, in this example a maximum of seven Boolean functions must be applied to HMMs response to achieve the desired performance as in Figure 3.7b. The time complexity of these Boolean combinations is negligible compared to that of computing the log-likelihood of the test sequences with HMM.

When using the IBC_{ALL} technique to repair the ROC curve belonging to the HMM trained with $N = 12$ states, the resulting curve $IBC_{ALL}(N = 12)$ dominates other ROC curves on the test set. This is because its test set performance exceeds its expected validation set performance, while the expected $IBC_{ALL}(N = 4)$ performance decreases on the validation set. Similarly, the expected validation performance of the LCR technique can be largely different from that of the test set as shown for $LCR(N = 12)$ curve. In general, the performance of both repairing techniques is less robust to variances between validation and test sets, since repairing relies on the shape of one ROC curve. In contrast, combining several ROC curves according to IBC_{ALL} technique is more robust to such variance since the interactions among all ROC curves are considered.

A closer look at ROC-based repairing techniques shows that IBC_{ALL} and LCR are complimentary. For instance, ROC curves repaired for $\lambda_{N=4}$, $IBC_{ALL}(N = 4)$ and $LCR(N = 4)$, cross in both validation and test sets. In general, IBC_{ALL} performs

well when the concavities are located in the bottom-left or in the top-right corners of the ROC space. IBC_{ALL} exploits the asymmetry in the ROC curve shape caused by an imbalanced variances at the head or tail of class distributions. The LCR technique is efficient for repairing concavities that are located close to non-major diagonal, since it only considers the shape of the ROC curve when negating the responses of the largest concavity. A higher level of AUC performance can therefore be achieved by using the MRROC to combine ROC curves repaired using the IBC_{ALL} and LCR techniques.

The attempt to combine the responses of the ROC curves repaired with the IBC_{ALL} or LCR techniques using IBC_{ALL}, was not successful due to the low level of robustness of the repairing techniques. Combination are based on the few points that define repaired ROC curve points on the validation set, which may be in different locations on the test set. In contrast, the direct combination of ROC curves according to BC_{ALL} or IBC_{ALL} techniques starts with existing thresholds on the validation set. Although ROC curves thresholds may change on the test set, these algorithms proceed by using neighboring thresholds.

3.6.2 Results with Synthetic and Real Data

Figure 3.8 shows the average AUC performance on the test sets versus the number of training blocks of a μ-HMM system where each HMM is trained with a different number of states ($N = 4, 8, 10$). Results are produced for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$, and for various training set sizes (50 to 450 symbols) and detector window sizes ($DW = 2, 4, 6$). The performance of the composite system obtained with the MRROC combination for the original HMM ROC curves, is compared to those obtained with the BC_{AND}, BC_{OR}, BC_{ALL}[8] and IBC_{ALL} techniques. The AUC performance of STIDE is also shown for reference. The reader is referred to Section IV.2 of Appendix IV for additional supporting results – $AUCH_{0.1}$ and tpr values at $fpr = 0.1$.

[8]For simplicity, BC_{ALL} is also used to indicate BCM_{ALL} when combining multiple ROC curves.

As shown in these figures, the BC_{ALL} can significantly improve the AUC over the MR-ROC in all the presented cases. The performance of the MRROC approaches that of BC_{ALL} technique when the training data becomes abundant for the problem at hand, or when test cases are easily detected (e.g., when classes are well separated). For example, this is shown in Figure 3.8a when the training block size grows beyond 150 sequences, even STIDE is able to achieve a high level of performance on this simple scenario (that is rarely encountered in real-world applications).

Although the BC_{AND} and BC_{OR} fusion were able to increase the performance over the MRROC, their performances is often significantly lower than that of the BC_{ALL}, as shown in Figure 3.8c and d. The significant increase in performance achieved with the BC_{OR} supports prior conclusion by Tao and Veldhuis (Tao and Veldhuis, 2008). The authors recommend using the OR Boolean fusion for detecting outliers in biometrics applications. However, this may not hold true when the number of positive samples is very limited.

The IBC_{ALL} technique provides the highest level of performance over all the range of conducted experiments. Since it includes the BC_{ALL} in its first iteration, the performance of IBC_{ALL} is lower bounded by that of the BC_{ALL}. Results indicate that IBC_{ALL} is most suitable for cases in which data are limited and imbalanced, as shown in the first blocks of Figures 3.8b, d, and f, (see additional results in Section IV.2 of Appendix IV).

The performance of IBC_{ALL} improves significantly over that of the BC_{ALL}. This is due to the iterative nature of the IBC_{ALL} that is able to repeatedly exploit the information residing in the inferior points, and hence select better combinations.

When the number of blocks for training is limited, the performance of the IBC_{ALL} technique is higher than other combination techniques. In such cases, each HMM trained with a different order provides diverse information by capturing different data structure, which allows to increase the performance of IBC_{ALL}. When the amount training data becomes abundant, the HMMs tend to achieve the same performance with less diversity in

their responses, which degrades the performance achieved by IBC_{ALL}. With the increase of detector window size[9], the likelihood of anomalous sequences becomes smaller at a faster rate than normal ones, and hence increases HMM detection rate (Khreich et al., 2009b). As a consequence, HMMs responses become less diverse yielding to a decrease of IBC_{ALL} performance. The impact of DW on performance is illustrated with the second and more complex scenario below.

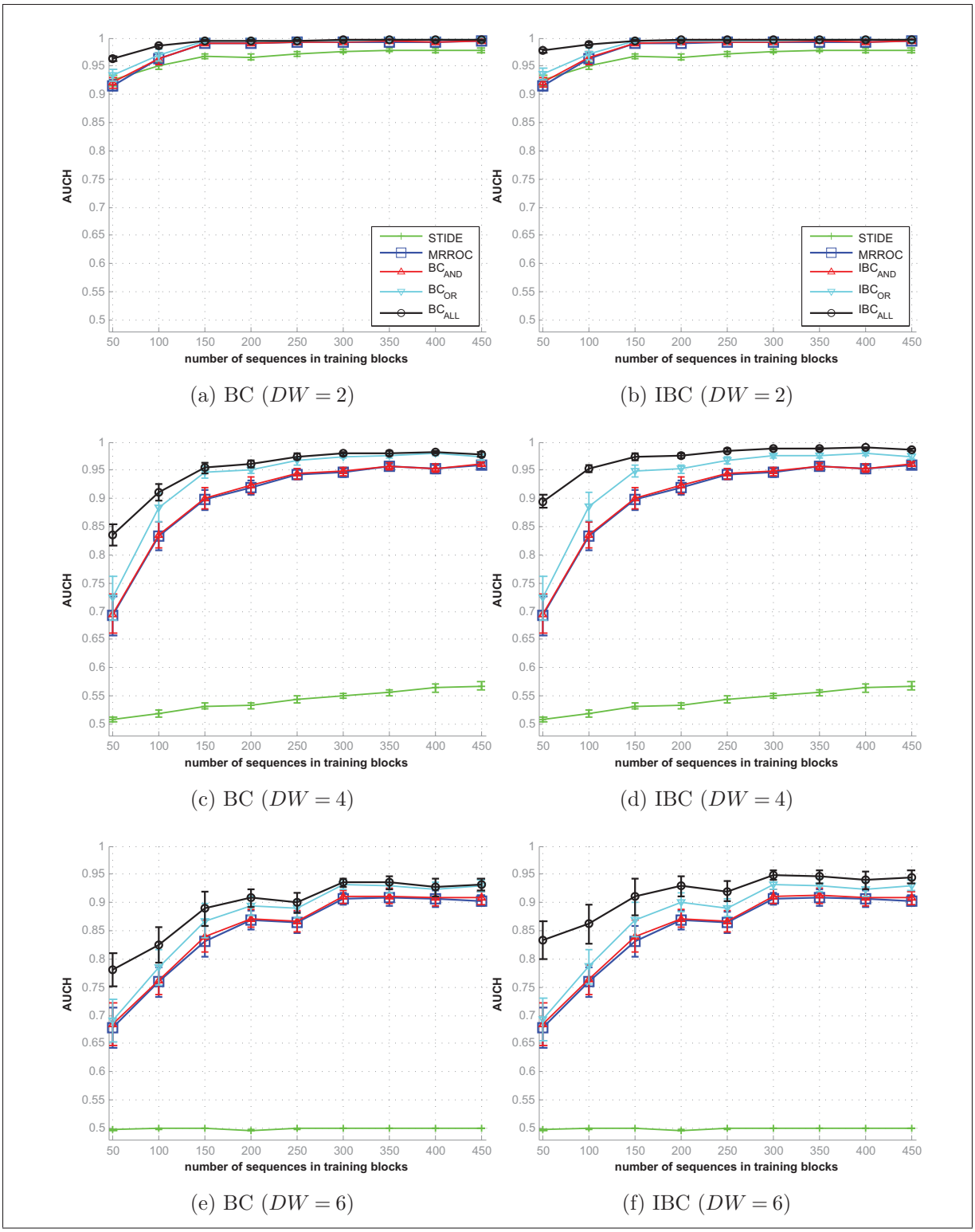
Since the IBC_{ALL} technique incorporates all combinations employed within the BC_{ALL} technique, only results of IBC_{ALL} are compared to those of MRROC, IBC_{AND} and IBC_{OR} for the second synthetic scenario ($\Sigma = 50, CRE = 0.4$) and for sendmail data.

Figure 3.9 confirms the results of Figure 3.8 on the second synthetic and more complex scenario, where $\Sigma = 50$ and $CRE = 0.4$. Again the IBC_{ALL} technique provides higher level of performance than the MRROC, IBC_{AND} and IBC_{OR} techniques. In this scenario, although the results of the AND and OR Boolean combinations were allowed to iterate until convergence, their achieved performances are still significantly lower than that of the IBC_{ALL}, as shown in Figure 3.9. This demonstrates the impact on performance of employing and iterating all Boolean functions until convergence. The performance gain is best illustrated for the first five training blocks (1000 to 3000 sequences), where the HMMs trained with different orders provide IBC_{ALL} with diverse responses for a significantly improved performance. Increasing the number of training blocks however increases the detection capabilities of the HMMs and reduces the diversity in their responses, which decreases the level of performance achieved with the IBC_{ALL} technique. As discussed previously, HMM detection ability increases with the detector window size (or anomaly size) which reduces the diversity in the μ-HMMs system. This negatively affects the performance achieved by the IBC_{ALL} technique as shown in Figure 3.9.

This is also confirmed on the sendmail data in Figure 3.10. Note however that with sendmail, the training data is very redundant and the test samples are very limited.

[9]For simplicity, the detector window size, DW, is assumed equal to the anomaly size, AS.

Even STIDE performance was moderate with only up to 1000 training sequences, out of the available 1.5 million sequences used for labeling. Nevertheless, the difference in performance between validation and test sets, not shown for improved visibility, is significantly large where the limited test samples are split in half for testing and half for validation. Although the IBC_{ALL} is able to increase the performance, this increase is not as prominent as in the synthetic cases. This is mainly due to redundancy in the training data, where the HMMs trained with different number of states were not able to capture enough diversity in the underlying data structure. Effective combination technique must exploit diverse and complimentary information to improve systems performance.

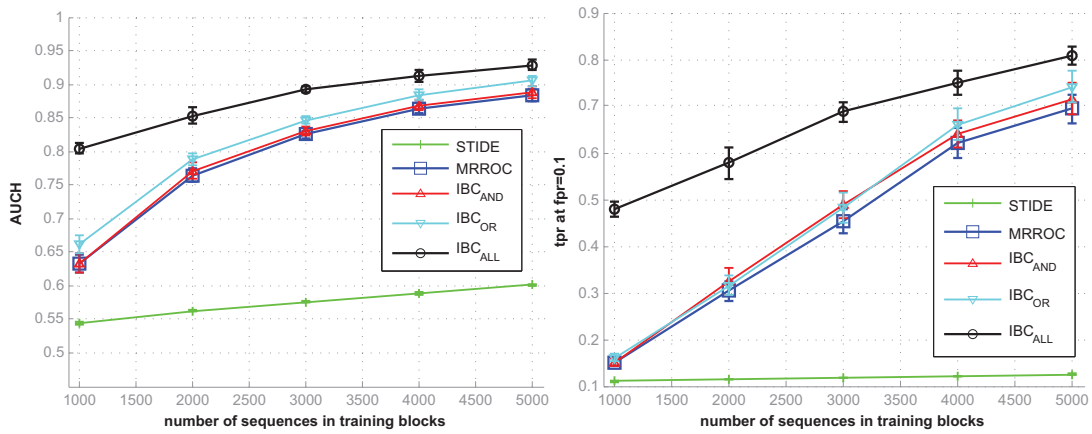


(a) BC ($DW = 2$) (b) IBC ($DW = 2$)

(c) BC ($DW = 4$) (d) IBC ($DW = 4$)

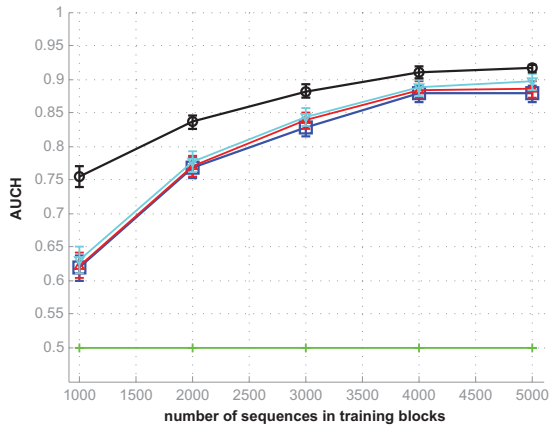
(e) BC ($DW = 6$) (f) IBC ($DW = 6$)

Figure 3.8 Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. Average AUC values obtained on the test sets as a function of the number of training blocks (50 to 450 sequences) for a 3-HMM system each trained with a different state ($N = 4, 8, 12$), and combined with the MRROC, BC and IBC techniques. Average AUC performance is compared for various detector windows sizes ($DW = 2, 4, 6$). Error bars are standard deviations over ten replications

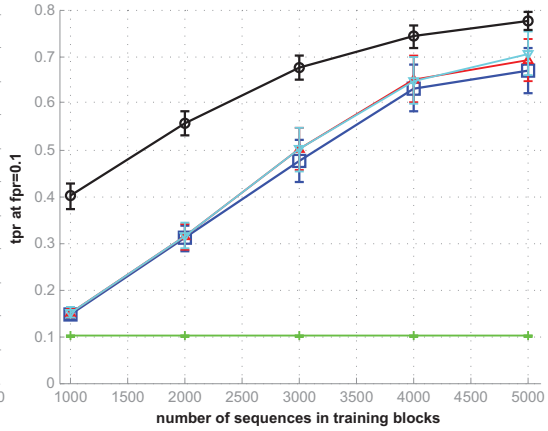


(a) DW=2

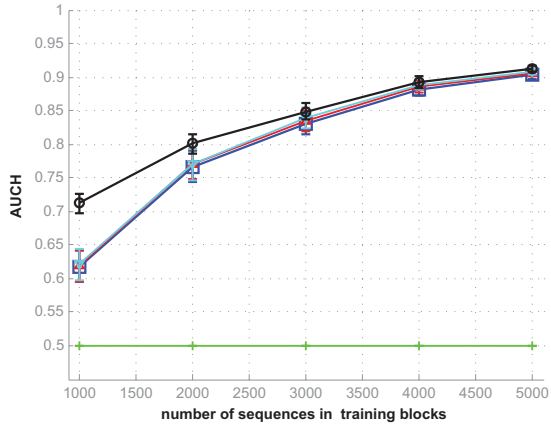
(b) DW=2



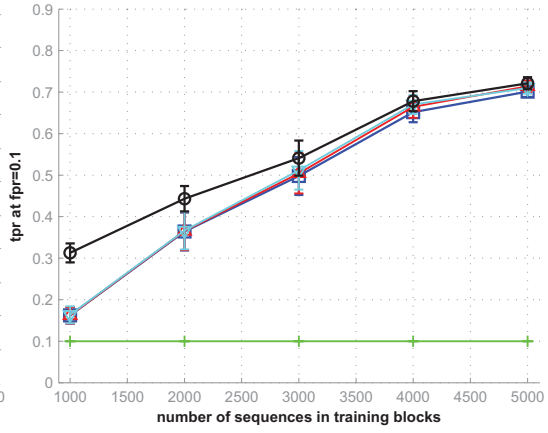
(c) DW=4



(d) DW=4

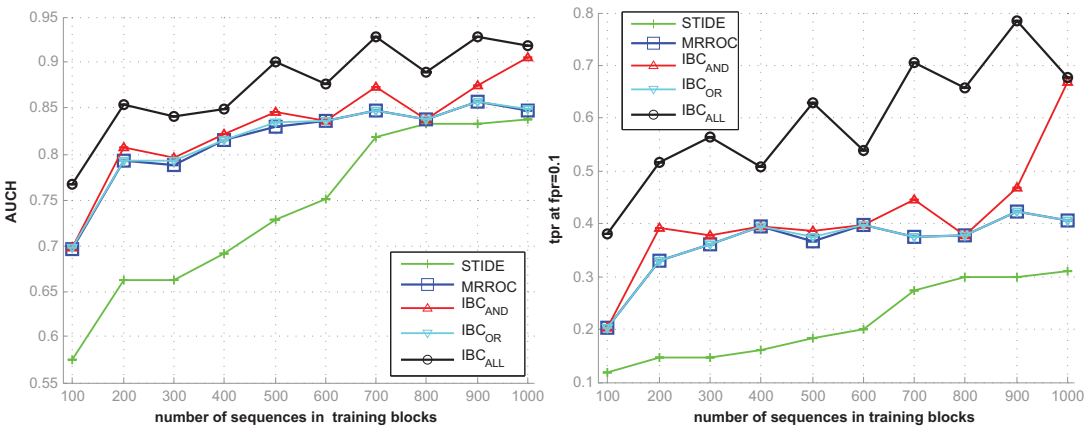


(e) DW=6



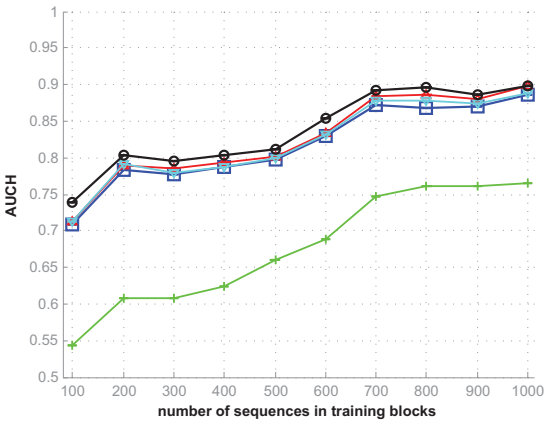
(f) DW=6

Figure 3.9 Results for synthetically generated data with $\Sigma = 50$ and $CRE = 0.4$. Average AUC values (left) and *tpr* values at $fpr = 0.1$ (right) obtained on the test sets as a function of the number of training blocks (1000 to 5000 sequences) for a 3-HMM system each trained with a different state ($N = 40, 50, 60$), and combined with the MRROC and *IBC* techniques. The performance is compared for various detector windows sizes ($DW = 2, 4, 6$). Error bars are standard deviations over ten replications

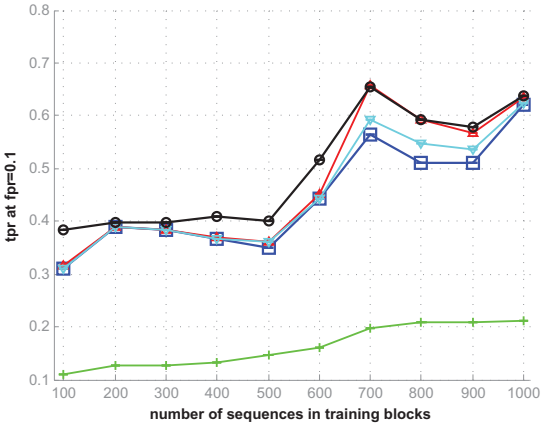


(a) DW=2

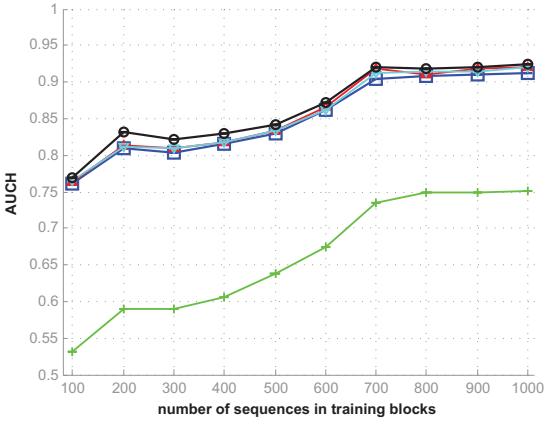
(b) DW=2



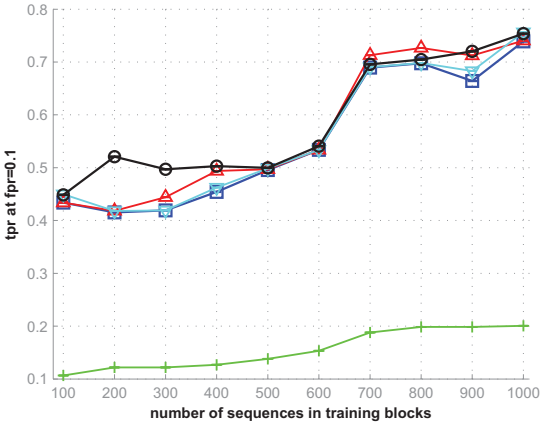
(c) DW=4



(d) DW=4



(e) DW=6



(f) DW=6

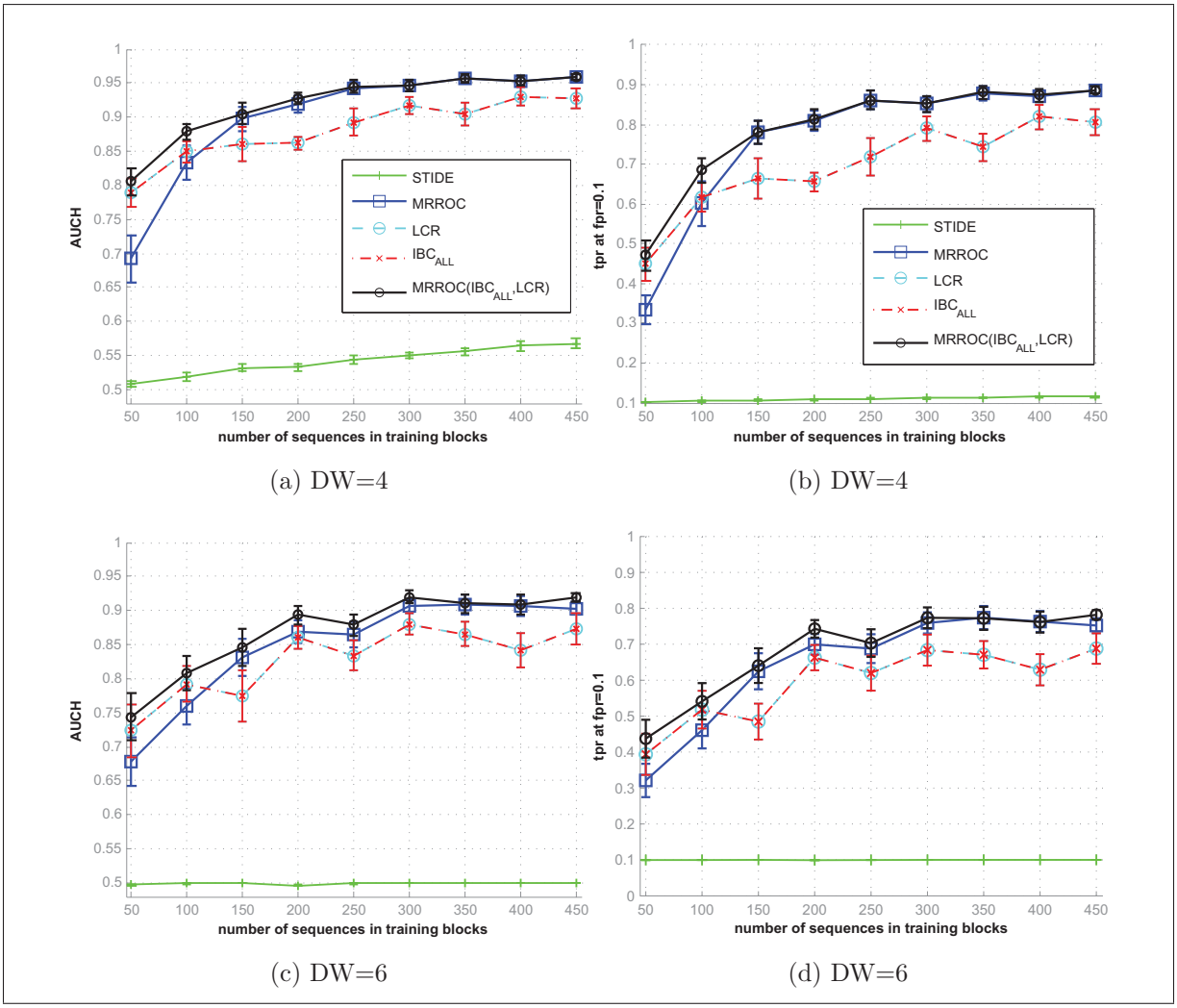
Figure 3.10 Results for sendmail data. AUC values (left) and *tpr* values at $fpr = 0.1$ (right) obtained on the test sets as function of the number of training blocks (100 to 1000 sequences) for a 5-HMM system, each trained with a different state ($N = 40, 45, 50, 55, 60$), and combined with the MRROC and IBC techniques. The performance is compared for various detector windows sizes ($DW = 2, 4, 6$)

Overall, the AUCs tend to increase to a comparable level as the number of training blocks increases. With a sufficient amount of training data, all HMMs are capable of achieving an equal level of performance, however with less diverse and complimentary information. On the other hand, the IBC_{ALL} performance will outperform others for small training set and detector window sizes. When the number of blocks for training is limited, the performance of the IBC_{ALL} technique is higher than other combination techniques. In such scenarios, each HMM trained with a different number of states is a able to capture different underlying structures of data. These HMMs provide diverse information, which allows to increase the performance of IBC_{ALL}. This holds true for different detector window sizes (DW), although the impact of combinations on performance degrades with the increase of DW values. HMMs are more capable of detecting larger anomalies, providing therefore less diverse responses to the IBC_{ALL} technique.

Increasing the number of HMMs trained with different orders (N) in the μ -HMM system has a significant impact on the performance achieved with the IBC_{ALL} technique due to the added diversity among the combined HMMs. Since diversity measures and the creation of diverse ensembles are not yet well defined in literature (Brown et al., 2005; Dos Santos et al., 2008), one can simply combine the responses of HMMs trained using a wide range of states to have an upper bound on performance. This is feasible in short period of time due to the efficiency of IBC_{ALL}. During operations however, simplified combination rules and fewer HMMs may be favored for speed constraints on the detection system. In such cases, the number of HMMs involved in the μ-HMM system can be reduced by selecting a subset of N values that does not significantly affect the upper bound achieved on performance (as described in Section 3.4.3). Training different HMMs on different subsets of the data provides other sources of diversity. Future work involves combining the responses of HMMs trained with different orders on different subsets of the data according to the IBC_{ALL} technique.

Finally, Figure 3.11 shows results for repairing concavities using the first synthetically generated scenario ($\Sigma = 8, CRE = 0.3$). The μ-HMM system is trained with three dif-

ferent states ($N = 4, 8, 12$), for various training set sizes and detector window sizes, and combined with the MRROC technique. Then, the IBC_{ALL} and LCR techniques are applied to repair the concavities presented in each ROC curve associated with an HMM. The repaired ROC curves are then combined according to the MRROC. In addition, the results of IBC_{ALL} and LCR techniques are also combined with the MRROC.



(a) DW=4 (b) DW=4

(c) DW=6 (d) DW=6

Figure 3.11 Performance versus the number of training blocks achieved after repairing concavities for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. Average AUCs (left) and *tpr* values at $fpr = 0.1$ (right) on the test set for a μ-HMM where each HMM is trained with different number of states ($N = 4, 8, 12$). HMMs are combined with the MRROC technique and compared to the performance of IBC_{ALL} and LCR repairing techniques, for various training block sizes (50 to 450 sequences) and detector windows sizes ($DW = 4$ and 6)

Figure 3.11 shows that each of the repairing techniques, IBC_{ALL} and LCR, is able to exceed the MRROC of the original curves at the first blocks. This is because in such cases the ROC curves comprise large concavities due to the limited training data. However, when the amount of training data increases, IBC_{ALL} and LCR are not able to provide a higher performance than the MRROC of the original curves unless their responses are themselves combined with the MRROC technique. This noticeable increase in performance, achieved by combining the repaired curves according to IBC_{ALL} and LCR with the MRROC, clearly indicates the complementarity of both techniques as discussed in Section 3.6.1.

However, the performance achieved by combining the repaired curves according to IBC_{ALL} and LCR with the MRROC is still lower than that of combining the original ROC curves using IBC_{ALL}, as presented in Figures 3.8d and f. Nevertheless, in results not shown in this chapter, repairing with the IBC_{ALL} and LCR techniques have shown lack of robustness to changes between the validation and test sets (as discussed in Section 3.6.1). In addition, repairing relies on large ROC concavities and they are not designed to improve the performance when the ROC curve comprises small concavities, but represent a poor performance (e.g., parallel to the diagonal of chance). In contrast, combining ROC curves using the IBC_{ALL} may allow to exploit this information to increase performance. In practice, when a ROC curve of a detector presents concavities, performance could be improved by combining the responses of both repairing techniques, IBC_{ALL} and LCR, using the MRROC fusion. Otherwise, the IBC_{ALL} could be directly applied to combine the responses of several ROC curves and provide a higher level of performance than other techniques.

3.7 Conclusion

This chapter presents an Iterative Boolean Combination (IBC) technique for efficient fusion of the responses from multiple classifiers in the ROC space. The IBC efficiently exploits *all* Boolean functions applied to the ROC curves and requires no prior assump-

tions about conditional independence of detectors or convexity of ROC curves. Although it seeks a sub-optimal set of combinations, the *IBC* is very efficient in practice and it provides a higher level of performance than related ROC-based combination techniques, especially when training data is limited and test data is heavily imbalanced. Its time complexity is linear with the number of classifiers while, the memory requirement is independent of the number of classifier, which allows for a large number of combinations. The proposed *IBC* is general in that it can be employed to combine diverse responses of any crisp or soft one- or two-class classifiers, within a wide range of application domains. This includes combining the responses of the same classifier trained on different data or features or trained according to different parameters, or from different classifiers trained on the same data, etc. It is also useful for repairing the concavities in a ROC curve.

During simulations conducted on both synthetic and real HIDS data sets, the *IBC* has been applied to combine the responses a of multiple-HMM system, where each HMM is trained using a different number of states, and capturing different temporal structures of the data. Results indicate that the *IBC* significantly improves the overall system performance over a wide range of training set sizes with various alphabet sizes and complexities of monitored processes, and according to different anomaly sizes, without a significant computational and storage overhead. Results have shown that, even with one iteration, the *IBC* technique always increases system performance over the MRROC fusion, and over the Boolean conjunction and disjunction combinations. When the *IBC* is allowed to iterate until convergence, the system performance improves significantly and the time and memory complexity required for each iteration are reduced by an order of magnitude with reference to the first iteration. The performance gain, especially when provided with limited training data, is due to the ability of the *IBC* technique to exploit diverse information residing in inferior points on the ROC curves, which are disregarded by the other techniques. In addition, repairing the concavities in a ROC curve using the *IBC* technique can yield higher level of performance than with the MRROC technique alone. The impact on performance of repairing the concavities is shown to be comparable and

complementary to an existing repairing technique that relies on inverting the largest concavity section. Therefore, combining the results of both repairing techniques according to the MRROC fusion yields a higher level of performance than applying each repairing technique alone.

In this chapter, the proposed ROC-based iterative Boolean combination technique is applied for fusion of responses from HMMs trained on fixed-size data sets. In next chapter, an adaptive system is proposed for incremental learning of new data over time using a learn-and-combine approach based on the proposed Boolean combination techniques.

CHAPTER 4

ADAPTIVE ROC-BASED ENSEMBLES OF HMMS APPLIED TO ANOMALY DETECTION*

In this chapter, an efficient system is proposed to accommodate new data using a learn-and-combine approach. When a new block of training data becomes available, a new pool of base HMMs is generated from the data using a different number of HMM states and random initializations. The responses from the newly-trained HMMs are then combined to those of the previously-trained HMMs in receiver operating characteristic (ROC) space using a novel incremental Boolean combination technique. The learn-and-combine approach allows to select a diversified ensemble of HMMs (EoHMMs) from the pool, and adapts the Boolean fusion functions and thresholds for improved performance, while it prunes redundant base HMMs. The proposed system is capable of changing its desired operating point during operations, and this point can be adjusted to changes in prior probabilities and costs of errors. Computer simulations conducted on synthetic and real-world host-based intrusion detection data indicate that the proposed system can achieve a higher level of performance than when parameters of a single best HMM are estimated, at each learning stage, using reference batch and incremental learning techniques. It also outperforms the static fusion functions (e.g., majority voting) for combining the new pool of HMMs with previously-generated HMMs. Over time, the proposed ensemble selection techniques have shown to form compact EoHMMs for operations, while maintaining or improving the overall system accuracy. Pruning has allowed to limit the pool size from increasing indefinitely, reducing thereby the storage space for accommodating HMMs parameters without negatively affecting the overall EoHMMs performance. Although applied for HMM-based ADSs, the proposed approach is general and can be employed for a wide range of classifiers and detection applications.

*THIS CHAPTER IS ACCEPTED, IN PRESS, IN PATTERN RECOGNITION JOURNAL, ON JULY 5, 2011, SUBMISSION NUMBER: PR-D-10-01131

4.1 Introduction

Anomaly detection systems (ADSs) detect intrusions by monitoring for significant deviations from normal system behavior. Traditional host-based ADSs monitor for significant deviation in normal operating system calls – the gateway between user and kernel mode. ADSs designed with discrete hidden Markov models (HMMs) have been shown to produce a high level of accuracy (Hoang and Hu, 2004; Warrender et al., 1999). A well trained HMM provides a compact detector that captures the underlying structure of a process based on the temporal order of system calls, and detects deviations from normal system call sequences with high accuracy and tolerance to noise.

An ADS allows to detect novel attacks, however will typically generate false alarms due in large part to the limited amount of representative data (Chandola et al., 2009a). Because the collection and analysis of training data to design and validate an ADS is costly, the anomaly detector will have an incomplete view of the normal process behavior, and hence misclassify rare normal events as anomalous. Substantial changes to the monitored environment, such as application upgrade or changes in users behavior, reduce the reliability of the detector as the internal model of normal behavior diverges with respect to the underlying data distribution. Since new training data may become available over time, an ADS should accommodate newly-acquired data, after it has originally been trained and deployed for operations, in order to maintain or improve system performance.

The incremental re-estimation of HMM parameters is a common approach to accommodate new data, although it raises several challenges. Parameters should be updated from new data without requiring access to the previous training data, and without corrupting previously-learned models of normal behavior (Grossberg, 1988; Polikar et al., 2001). Standard techniques for estimating HMM parameters involve iterative batch learning algorithms (Baum et al., 1970; Levinson et al., 1983), and hence require observing the entire training data prior to updating HMM parameters. Given new training data, these techniques can be costly since they involve restarting the HMM training using all cu-

mulative training data. Alternatively, several efficient on-line learning techniques have been proposed to re-estimate HMM parameters continuously upon observing each new observation symbol or new observation sub-sequence from an infinite data stream (Florez-Larrahondo et al., 2005; Mizuno et al., 2000). In practice however, on-line learning using blocks comprising limited amount of data yields a low level of performance as one pass over each block is not sufficient to capture the phenomena (Khreich et al., 2009a). It is also possible to adapt on-line learning techniques for incremental learning over several passes of each block, although it requires strategies to manage the learning rate (Khreich et al., 2009a). Moreover, using a single HMM with a fixed number of states for incremental learning may not capture a representative approximation of the underlying data distribution due to the many local maxima in the solution space.

Ensemble methods have been recently employed to overcome the limitations faced with a single classifier system (Dietterich, 2000; Kuncheva, 2004a; Polikar, 2006; Tulyakov et al., 2008). Theoretical and empirical evidence have shown that combining the outputs of several accurate and diverse classifiers is an effective technique for improving the overall system accuracy (Brown et al., 2005; Dietterich, 2000; Kittler, 1998; Kuncheva, 2004a; Polikar, 2006; Rokach, 2010; Tulyakov et al., 2008). In general, designing an ensemble of classifiers involves generating a diverse pool of base classifiers (Breiman, 1996; Freund and Schapire, 1996; Ho et al., 1994), selecting an accurate and diversified subset of classifiers (Tsoumakas et al., 2009), and then combining their output predictions (Kuncheva, 2004a; Tulyakov et al., 2008). Most existing ensemble techniques aim at maximizing the overall ensemble accuracy, which assumes fixed prior probabilities and fixed misclassification costs. In many real world applications, like anomaly detection, prior class distributions are highly imbalanced and misclassification costs may vary over time. Nevertheless, common techniques used for fusion, such as voting, sum or averaging, assume independent classifiers and do not consider class priors (Kittler, 1998; Kuncheva, 2002). This chapter focuses on HMM-based systems applied to anomaly detection, and

in particular on the efficient adaptation of ensembles of HMMs (EoHMMs) over time, in response to new data.

The receiver operating characteristic (ROC) curve is commonly used for evaluating the performance of detectors at different operating points, without committing to a single decision threshold (Fawcett, 2004). ROC analysis is robust to imprecise class distributions and misclassification costs (Provost and Fawcett, 2001). In a previous work, the authors proposed a ROC-based iterative Boolean combination (IBC) technique for fusion of responses from any crisp or soft detector trained on *fixed-size* data sets (Khreich et al., 2010b). The IBC algorithm inputs *all* generated classifiers, and combines their responses in the ROC space by applying all Boolean functions, over several iterations until the ROC convex hull (ROCCH) stops improving. IBC was applied to combine the responses from a multiple-HMM ADS, where each HMM is trained through batch learning using the *same* data but with different number of states and initializations. Results have shown that detection accuracy improves significantly, especially when training data are limited and class distributions are imbalanced (Khreich et al., 2010b). The IBC is a general decision-level fusion technique in the ROC space that aims at improving ensembles accuracy, when learning is performed from a fixed-size set of limited training data. However, IBC does not allow to efficiently adapt a fusion function over time when new data becomes available, since it requires a fixed number of classifiers.

In contrast, this chapter presents a new ROC-based system to efficiently adapt EoHMMs over time, from new training data, according to a learn-and-combine approach. Given new training data, a new pool of HMMs is generated from newly-acquired data using different HMM states and initializations. The responses from these newly-trained HMMs are then combined with those of the previously-trained HMMs in ROC space using the *incremental* Boolean combination ($incrBC$) technique. $IncrBC$ extends on IBC in that Boolean fusion of HMMs may be adapted for new data, without multiple iterations. However, system efficiency and scalability remain a serious issue. On its own, $incrBC$ allows to discard previously-learned data, yet it retains all previously-generated HMMs in

the pool. Over time, the pool size would therefore grow indefinitely, yielding unnecessarily high memory requirements, and increasingly complex Boolean fusion rules. Operating ever-growing EoHMMs would increase computational and memory complexity.

To overcome these limitations, specialized algorithms are proposed for memory management. First, ensemble selection algorithms are proposed to efficiently select diversified EoHMMs from the pool, and to adapt the Boolean fusion functions and decision thresholds to improve overall system performance. The proposed system may employ one of two ensemble selection algorithms, called BC_{greedy} and BC_{search}, that are adapted to benefit from the monotonicity in $incrBC$ accuracy, for a reduced complexity. Both algorithms feature rank- and search-based selection strategies to optimize ROOCH of combinations, and hence maximize the area under the ROCCH (AUCH). Both algorithms preserve the ROCCH of combination which allows visualizing the expected performance over the entire ROC space, and adapting to changes in operating conditions during system operations, such as tolerated false alarm rate, prior probabilities and costs of errors. Most existing techniques for adapting ensembles of classifiers require restarting the training, selection, combination, and optimization procedures to account for such a change in operating conditions. Finally, to address the potentially growing memory requirements over time, the proposed system integrates a memory management strategy to prune less relevant HMMs from the pool.

For a proof-of-concept validation, this system is applied to adaptive anomaly detection from system call sequences. The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets[1] (Warrender et al., 1999). UNM data sets are still commonly used in literature (Forrest et al., 2008; Kershaw et al., 2011) due to limited publicly available system call data sets. In fact labeling real system call sequences is a challenging task, which depends on detector window size. To overcome issues encountered when using real-world system call data for anomaly-based HIDS, the experimental results are complemented with synthetically gen-

[1] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

erated data. The synthetic data generator is based on the Conditional Relative Entropy (CRE), and is closely related to the work of Tan and Maxion (Tan and Maxion, 2003). It allows simulating different processes with various complexities and provides the desired amount of normal data for training and labeled data (normal and anomalous) for testing. For both real and synthetic data sets, the data are organized into several blocks that are assumed to have been acquired over time. The new data allow to account for rare events, and adapt to changes in monitored environments such as application updates or changes in user behavior over time. The performance of the proposed system is compared to that of the reference batch BW (BBW) trained on all cumulative data, of on-line BW (OBW) (Mizuno et al., 2000), and of an algorithm for incremental learning of HMM parameters based on BW (IBW) BW (IBW) (Khreich et al., 2009a) described in Appendix III. The performance of the median and majority vote fusion functions, combining outputs from the previously-generated pool of HMMs, are also presented. Accuracy is assessed using the area under the ROC curve (AUC) (Hanley and McNeil, 1982) and the true positive rate (tpr) achieved at a fixed false positive rate (fpr) value.

The rest of this chapter is organized as follows. The next section briefly reviews the HMM, as it applies to adaptive ADS from system call sequences. It also reviews techniques for incremental learning of HMM parameters, and for designing multiple classifiers systems. Section 4.3 describes the proposed ROC-based system for efficient adaptation of EoHMM from new data according to the learn-and-combine approach, including new techniques for Boolean combination and model management. Section 4.4 presents the experimental methodology (data sets, evaluation methods and performance metrics) used for proof-of-concept computer simulations. Simulation results are presented and discussed in Section 4.5.

4.2 Adaptive Anomaly Detection Systems

4.2.1 Anomaly Detection Using HMMs

HMMs have been shown successful in detecting anomalies in sequences of operating system calls (Chen and Chen, 2009; Hoang and Hu, 2004; Khreich et al., 2009b; Tan and Maxion, 2003; Warrender et al., 1999). A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, N, and a set of observation probability distributions, each one associated with a state. Starting from an initial state $S_i \in \{S_1, ..., S_N\}$, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij}. The process then emits a symbol v_k, from a finite alphabet $V = \{v_1, \ldots, v_M\}$ with M distinct observable symbols, according to the discrete-output probability distribution $b_j(v_k)$ of the current state S_j. An HMM is commonly parametrized by $\lambda = (\pi, A, B)$, where the vector $\pi = \{\pi_i\}$ is the initial state probability distribution, matrix $A = \{a_{ij}\}$ is the state transition probability distribution, and matrix $B = \{b_j(v_k)\}$ is the state output probability distribution, ($1 \leq i, j \leq N$ and $1 \leq k \leq M$).

HMMs can perform three canonical functions, evaluation, decoding and learning (Rabiner, 1989). Evaluation aims to compute the likelihood of an observation sequence $o_{1:T}$ given a trained model λ, $P(o_{1:T} \mid \lambda)$. The likelihood is typically evaluated by using a fixed-interval smoothing algorithm such as the Forward-Backward (FB) (Rabiner, 1989) or the numerically more stable Forward-Filtering Backward-Smoothing (FFBS) (Ephraim and Merhav, 2002). Decoding means finding the most likely state sequence S that best explains a given observation sequence $o_{1:T}$ (i.e, maximize $P(S \mid o_{1:T}, \lambda)$). The best state sequence is commonly determined by the Viterbi algorithm. Learning aims to estimate the HMM parameters λ to best fit the observed batch of data $o_{1:T}$. HMM parameters estimation is frequently performed according to the maximum likelihood estimation (MLE) criterion. MLE consists of maximizing the log-likelihood, $\log P(o_{1:T} \mid \lambda)$, of the train-

ing data over HMM parameters space. Unfortunately, since the log-likelihood depends on missing information (the latent states), there is no known analytical solution to the learning problem. In practice, iterative optimization techniques such as the Baum-Welch (BW) algorithm (Baum et al., 1970), a special case of the Expectation-Maximization (EM) (Dempster et al., 1977), or standard numerical optimization methods such as gradient descent (Baldi and Chauvin, 1994; Levinson et al., 1983) are often employed for this task. In either case, HMM parameters are estimated over several training iterations, until the likelihood function is maximized over data samples. Each training iteration typically involves observing all available training data to evaluate the log-likelihood value and estimate the state densities by using a fixed-interval smoothing algorithm.

Anomaly detection creates a profile that describes normal behaviors. Events that deviate significantly from this profile are considered anomalous. In computer and network security, legitimate system call sequences generated during the normal execution of a privileged process are typically employed in host-based ADSs (Chandola et al., 2009a; Warrender et al., 1999). As stochastic models for temporal data, HMMs provide compact detectors that are able to capture probability distributions corresponding to complex real-world phenomena, with tolerance to noise and uncertainty. Given a sufficiently large training set of legitimate system call observations, an ergodic[2] HMM trained according to the BW or gradient-based algorithms, is able to capture the underlying structure of the monitored process (Hoang and Hu, 2004; Warrender et al., 1999). During operations, system calls sequences generated by the monitored process are evaluated with the trained HMM, according to the FB or FFBS algorithms, which should assign significantly lower likelihood values to anomalous sequences than to normal ones. By setting a decision threshold on the likelihood values, monitored sequences can be classified as normal or anomalous.

[2]An HMM with an ergodic or fully connected topology is typically employed in situations where the hidden states have no physical meaning such as modeling a process behavior using its generated system calls.

Estimating the parameters of a HMM requires the specification of the number of hidden states. Although it is a critical parameter for HMM performance, this number is often chosen heuristically or empirically, by selecting the single value that provides the best performance on training data (Chen and Chen, 2009; Hoang and Hu, 2004; Warrender et al., 1999). Using a single HMM with a pre-specified number of states may have limited capabilities to capture the underlying structure of the data. HMMs trained with a fixed-size training data and with a different number of states and initializations have been shown to provided a higher level of performance in host-based ADSs (Khreich et al., 2009b).

4.2.2 Adaptation in Anomaly Detection

The design of accurate detectors for ADS requires the collection of a sufficient amount of representative data. In practice however, it is very difficult to collect, analyze and label comprehensive data sets for reasons that range from technical to ethical. Limited normal data are typically provided for training HMM-based detectors, and limited labeled data are also provided for validation of an ADS. Furthermore, the underlying data distribution of normal behaviors may vary according to gradual, systematic, or abrupt changes in the monitored environment such as an application update or changes in users behavior. The ability to incrementally adapt HMM parameters in response to newly-acquired training data from the operational environment or other sources is therefore an undisputed asset for sustaining a high level of performance. Indeed, refining a HMM to novelty encountered in the environment may reduce its uncertainty with respect to the underlying data distribution.

Assume that an HMM-based ADS is a priori trained, optimized and validated off-line using a finite set of system call data, to provide an acceptable level of performance in terms of false and true positive rates. During operations, monitoring a centralized server, for instance, the system is susceptible to produce a higher false alarms rate than tolerated by the system administrator. This is largely due to the limited amount data

that is available for training. The anomaly detector will have an incomplete view of the normal process behavior, and rare events will be mistakenly considered as anomalous. The HMM detector should be refined incrementally on some additional normal data when it becomes available, to better fit the normal behavior of process in consideration. Such situations may also occur when trying to implement an HMM-based ADS specialized for monitoring a specific process, but on different hosts machines with different settings and functionality. Generic models that are trained using data corresponding to common settings, could be set into operation then incrementally refined to fit the normal process behavior on each host. Otherwise, training data must be collected and analyzed from the same process on every host.

As a part of an ADS, the system administrator plays a crucial role in providing new data for training and validation of the detectors. By accommodating new data and refining its detection boundaries ADSs motivate the administrator to interact more positively and build confidence in the system responses over time. When an alarm is raised, the suspicious system call sub-sequences are logged and analyzed for evidence of an attack. If an intrusion attempt is confirmed, the corresponding anomalous system calls should be provided to update the validation set ($\mathcal{V}$). A response team should react to limit the damage, and a forensic analysis team should investigate the cause of the successful attack. Otherwise, the intrusion attempt is considered as a false alarm and these rare sub-sequences are tagged as normal and employed to update the HMM detectors. One challenge is the efficient integration of the newly-acquired data into the ADS without corrupting the existing knowledge structure, and thereby degrading the performance.

4.2.3 Techniques for Incremental Learning of HMM Parameters

Incremental learning refers to the ability of an algorithm to learn from new data after a classifier has already been trained from previous data and deployed for operations. As illustrated in Figure 4.1, once a new block of data becomes available, incremental learning produces a new hypothesis (i.e., decision rule) that depends only on the previous

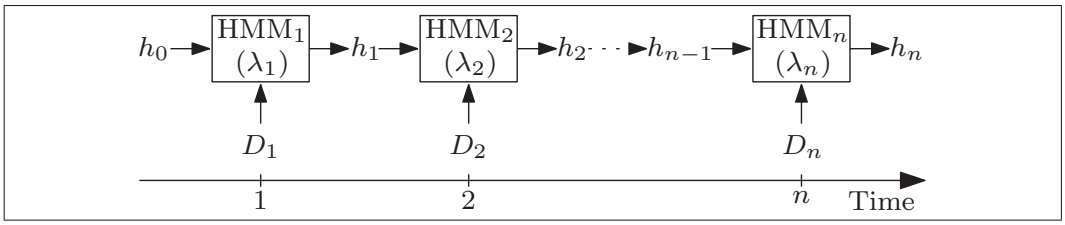


Figure 4.1 An incremental learning scenario where blocks of data are used to update the HMM parameters (λ) over a period of time. Let $D_1, D_2, ..., D_n$ be the blocks of training data available to the HMM at discrete instants in time. The HMM starts with an initial hypothesis h_0 associated with the initial set of parameters λ_0, which constitutes the prior knowledge of the domain. Thus, h_0 and λ_0 get updated to h_1 and λ_1 on the basis of D_1, then h_1 and λ_1 get updated to h_2 and λ_2 on the basis of D_2, and so forth

hypothesis and the current training data (Polikar et al., 2001). Incremental learning allows to decrease the memory requirements needed to learn the new data, since there is no need to store and access previously-learned data. Furthermore, since training is performed only on new training blocks, and not on all accumulated data, incremental learning would also lower the time complexity needed to learn new data.

The main challenge of incremental learning is the ability to sustain a high level of performance without corrupting previously-learned knowledge (Grossberg, 1988; Polikar et al., 2001). In fact, considering the HMM parameters obtained using a batch learning technique on a first block of sub-sequences as starting point for training the HMM on a second newly-acquired block, may not allow the optimization process to escape the local maximum associated with the first block. Remaining trapped in local maxima leads to knowledge corruption and hence to a decline in system performance (see Figure 4.2).

Several on-line learning techniques have been proposed in literature to re-estimate HMM parameters from a continuous stream of symbols. The objective is to optimize HMM parameters to fit the source generating the data through one observation of the data. These techniques are typically designed to reduce the training time and memory complexity and to accelerate the convergence when the learning scenario involves very long stream of data. These techniques are derived from their batch counterparts, and in-

clude, for instance, EM-based (Florez-Larrahondo et al., 2005; Mizuno et al., 2000) and gradient-based (Baldi and Chauvin, 1994; LeGland and Mevel, 1997) techniques. However, the key difference is that the on-line optimization and update of HMM parameters are continuously performed upon observing each new sub-sequence (Baldi and Chauvin, 1994; Mizuno et al., 2000) or new symbol (Florez-Larrahondo et al., 2005; LeGland and Mevel, 1997) of observation, with no iterations.

In practice however, when an on-line learning technique is applied to incremental learning of a finite data set, one pass over the sub-sequences within a new training block is insufficient to capture the phenomena. In addition, the convergence properties of these on-line algorithms no longer hold when provided with limited amount of data. Several iterations over the new block of observations are therefore required to better fit the HMM parameters to the newly-acquired data, and hence provide an operational model with acceptable performance. When several iterations are allowed, on-line algorithms may attain a local maximum on the log-likelihood function associated with the current block of data, and must overcome the same issue of remaining trapped in the previous local maximum (see Figure 4.2).

A potential advantage of applying on-line learning over batch learning techniques to incremental learning resides in their added stochasticity, which stems from the rapid re-estimation of HMM parameters after each sub-sequence or symbol of observations. This may aid escaping local maxima during the early adaptation to newly provided blocks of data. In addition, a fixed or monotonically decreasing learning rate is typically employed in these algorithms to integrate pre-existing knowledge of the HMM and the newly-acquired information associated with each sub-sequence or observation symbol. This may alleviate knowledge corruption.

Figure 4.2 illustrates the decline in performance that may occur with batch or on-line algorithms during incremental update of one HMM parameter. The values λ_1^*, λ_2^*, and λ_{12}^* correspond to the optimal values of parameters after learning D_1, D_2 and $D_1 \cup D_2$,

respectively. Assume that the training block D_2, which contains one or multiple subsequences, becomes available after an HMM(λ_1) has been previously trained on a block D_1 and deployed for operations. An incremental algorithm that optimizes, for instance, the log-likelihood function must re-estimate the HMM parameters over several iterations until this function is maximized for D_2. The optimization of HMM parameters depends on the shape and position of the log-likelihood function of HMM(λ_2) with respect to that of HMM(λ_1). When an incremental learning technique is employed to train on D_2 alone, λ_1^* is the starting point for re-estimating the HMM. If the log-likelihood function of HMM(λ_2) has some local maxima in the proximity of λ_1^*, the optimization may remain trapped in these maxima, and hence does not accommodate D_2, providing a similar model parameter. If λ_1^* was selected according to point (a), the optimization will become trapped in the local maximum at point (c) instead of approaching λ_{12}^*. This leads to a knowledge corruption, and a decline in HMM performance. In contrast, training a new HMM(λ_2) on D_2 from the start using different initializations may lead to point (d). As described in subsequent sections, combination of responses from two HMMs selected according to λ_1^* and λ_2^* exploits their complementary information for a higher overall level of performance.

4.2.4 Incremental Learning with Ensembles of Classifiers

Multiple classifier systems (MCSs) are based on the combination of several accurate and diverse base classifiers to improve the overall system accuracy (Brown et al., 2005; Dietterich, 2000; Kittler, 1998; Kuncheva, 2004a). MCSs have been employed to overcome the limitations faced with a single classifier system (Dietterich, 2000; Kuncheva, 2004a; Polikar, 2006; Tulyakov et al., 2008). According to the no-free-lunch theorem (Wolpert and Macready, 1997), no algorithm is best for all possible problems. Classifiers may converge to different local optima according to different initializations or different parameter values. Different classifiers can have different domains of expertise or perceptions of the same problem. Therefore, combining the predictions of an ensemble of classifiers (EoC) reduces the risk of selecting a single classifier with poor generalization performance.

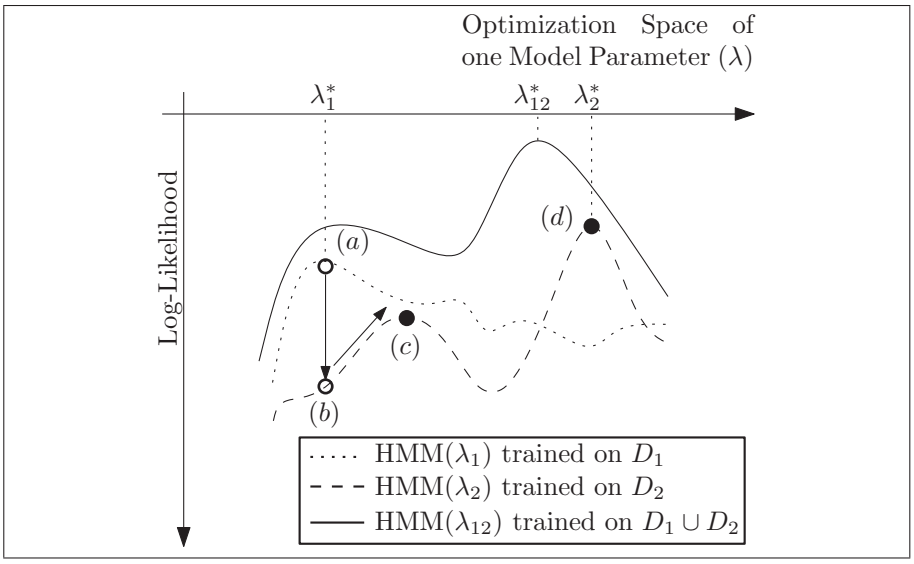


Figure 4.2 An illustration of the decline in performance that may occur with batch or on-line estimation of HMM parameters, when learning is performed incrementally on successive blocks

Furthermore, base classifiers may commit different errors providing an EoC with complementary information that increase system performance. EoCs provide an alternative solution for incremental learning by combining classifiers trained on each block of data as it becomes available, instead of selecting and updating a single classifier.

In general, the design of EoCs involves generating a diversified pool of base classifiers, selecting a subset of members from that pool, and then combining their output predictions such that the overall accuracy and reliability is improved. A pool can be composed of either homogeneous or heterogeneous classifiers. Homogeneous classifiers are generated by the same classifier trained using different manipulations of parameters, training data (Breiman, 1996; Freund and Schapire, 1996), or input features (Ho, 1998). Heterogeneous classifiers are generated by training different classifiers on the same data set.

The combination of selected classifiers typically occurs at the score, rank or decision levels (Kuncheva, 2004a; Tulyakov et al., 2008), and may be static (e.g., sum, product, majority voting, etc.), adaptive (e.g., weighted average, weighted voting, etc.) or trainable (also known as stacked or meta-classifier). Fusion at the score level is most

prevalent in literature (Kittler, 1998). Normalization of the scores is typically required before applying static or adaptive fusion functions, which may not be a trivial task for heterogeneous classifiers. Trainable approaches, where a global meta-classifier is trained for fusion on classifiers responses (Kittler, 1998; Roli et al., 2002; Ruta and Gabrys, 2005; Wolpert, 1992), are prone to overfitting and require a large independent validation set for estimating the parameters of the combining classifier (Roli et al., 2002; Wolpert, 1992). Fusion at the rank level is mostly suitable for multi-class classification problems, where the correct class is expected to appear in the top of the ranked list (Ho et al., 1994; Van Erp and Schomaker, 2000). Fusion at the decision level exploits the least amount of information since only class labels are input to the combiner. Compared to the other fusion methods, it is less investigated in literature. Majority voting (Ruta and Gabrys, 2002), weighted majority voting (Ali and Pazzani, 1996; Littlestone and Warmuth, 1994) and behavior-knowledge-space (BKS) methods (Raudys and Roli, 2003) are the most representative decision-level fusing methods. One issue that appears with decision level fusion is the possibility of ties. The number of base classifiers must therefore be greater than the number of output classes. BKS only applies to low dimensional problems. Moreover, in order to have an accurate probability estimation, it requires a large independent training and validation set to design the combination rules.

EoCs may be adapted for incremental learning by generating a new pool of classifiers as each block of data becomes available, and combining the outputs with those of previously-generated classifiers with some fusion technique (Kuncheva, 2004b). The main advantage of using EoCs to implement a learn-and-combine approach is the possibility of avoiding knowledge corruption as classifiers are trained from the start on a new block of data.

For instance, the adaptive boosting methods employed in AdaBoost (Freund and Schapire, 1996) have been extended for incremental learning from new blocks of data in Learn++ (Polikar et al., 2001). Given a fixed-size training set, AdaBoost iteratively generates a sequence of weak classifiers each focusing on training observations that have been misclassified by the previous classifier. This is achieved by updating a weight distribution

over the entire training set according to the performance of the previously-generated classifier. At each iteration, AdaBoost increases the weights of wrongly classified training observations and decreases those of correctly classified observations, such that the new classifier focuses on hard-to-classify observations. The final output is a weighted majority voting of all generated classifiers.

By contrast, Learn++ generates a number of base classifiers for each block of data that becomes available. The weight distribution is updated according to the performance of all previously-generated classifiers, which allows to accommodate previously unseen classes (Polikar et al., 2001). The outputs are then combined through weighted majority voting to obtain the final classification. Improvements proposed for this algorithm essentially differ by the combination functions and weight distribution update. In particular, Learn++ variant for imbalanced data (Muhlbaier et al., 2004) employs a class conditional weight factor for updating classifiers weight and accounting for class imbalance. This factor is the ratio of the number of observations from a particular class used for training a classifier, to the number of observations from the same class used for training all other previously-generated classifiers.

An on-line version of bagging and boosting ensembles have been proposed for learning from labeled streams of data (Oza and Russell, 2001). The bootstrap sampling and weight distribution update that are employed for fixed-sized data sets are now simulated according to Poisson distribution for data streams. For on-line AdaBoost algorithm, the ensemble is composed of a fixed number of classifiers. Each new observation is presented sequentially to the ensemble members. A classifier is trained on this observation k times, where k is drawn from a Poisson distribution parametrized by the observation weight. When an observation is misclassified its weight increases and hence will be presented more often to the following classifier in the ensemble. The final output is the weighted majority vote over all the base classifiers. However, on-line bagging and boosting techniques consider learning continuously from labeled streams of data.

The techniques described above are designed for two- or multi-class classification and hence require labeled training data sets to compute the errors committed by the base classifiers and update the weight distribution at each iteration. In HMM-based ADSs, the HMM detector is a one-class classifier that is trained using the *normal* system call data only as described in Sections 4.2.1 and 4.2.2. Therefore, they are not suitable for ADSs using system call sequences. Some authors however, adopt these techniques by considering that rare normal system call sequences are anomalous. For instance, an EoHMMs based on discrete AdaBoost and its on-line version has been applied for anomaly detection (Chen and Chen, 2009). With this method, an arbitrary threshold is set according to the log-likelihood output from HMMs trained on the normal system call behavior, below which normal system call sequences are considered as anomalous. Five HMMs are trained with the same number of states *on fixed-sized data*, and then employed as base detectors in AdaBoost algorithm. When new data becomes available, these HMMs are updated according to an on-line AdaBoost variant (Oza and Russell, 2001). Threshold setting is shown to have a significant impact on the detection performance of the EoHMMs (Chen and Chen, 2009). In fact, rare system call events are normal, if they are considered anomalous during the design phase, they will generate false alarms during the testing phase. These rare events may be suspicious if, during operation, they occur in bursts over a short period of time.

Another specific combination technique for HMMs consists of weight averaging the parameters of an HMM trained on a newly-acquired block of data with those of a previously-trained HMMs (Hoang and Hu, 2004). Although this technique may work for left-right HMMs, it is not suitable for ergodic HMMs as their states are permuted with each different training phase, and hence averaging parameters leads to knowledge corruption. Furthermore, it is restricted to combining HMMs with the same number of states.

For one- or two-class classification problems, Boolean combination of classifier responses in the ROC space provides several advantages over related fusion techniques (Khreich et al., 2010b; Tao and Veldhuis, 2008). It does not require any prior assumption regard-

ing the independence of classifiers, in contrast with other fusion functions such as sum, product or averaging that are based on the independence of classifiers (Kittler, 1998; Kuncheva, 2002). In addition, normalization of heterogeneous classifiers output scores is not required, because ROC curves are invariant to monotonic transformation of classification thresholds (Fawcett, 2004). Boolean combination techniques may therefore be applied to combine the responses from any crisp or soft homogeneous or heterogeneous classifiers, or from classifiers trained on different data. Most existing ensemble techniques, especially those proposed for incremental learning, aim at maximizing the system performance through a single measure of accuracy. This implicitly assumes fixed prior probabilities and misclassification costs. Furthermore, an arbitrary and fixed threshold is typically set (implicitly or explicitly) on the output scores of soft classifiers of an ensemble prior to taking a majority voting decision. In anomaly detection however, prior class distributions are highly imbalanced and misclassification costs are different and both may change over time. As detailed in Section 4.3.1, Boolean combination in the ROC space aims at maximizing the overall convex hull of combinations overall classifiers decision thresholds, which allows to adapt to changes in prior probabilities and cost of errors during system operation.

4.3 Learn-and-Combine Approach Using Incremental Boolean Combination

In this section, an adaptive system is proposed for incremental learning of new data over time using a learn-and-combine approach. As new blocks of data become available, the system learn new HMMs to evolve a pool of classifiers, and performs incremental Boolean combination ($incrBC$) of a diversified subset of HMM responses in the ROC space. This is achieved by selecting the HMMs, decision thresholds, and Boolean functions such that the overall EoHMMs performance is maximized.

The main components of the system are illustrated in Figure 4.3 and described in Algorithm 4.1. When a new block of *normal* training sub-sequences (D_k) becomes available, a new pool of base HMMs ($\mathcal{P}_k$) is generated according to different number of states and

random initializations. (Further details on HMMs training and validation are given in Section 4.4.2.) $\mathcal{P}_k$ is then appended to the previously-generated pool of base HMMs $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{k-1}\} \cup \mathcal{P}_k$, and D_k is discarded. The model management module, described in Section 4.3.2, performs model selection and pruning. The selection module forms the EoHMMs (E_k) considered for operations, by selecting HMMs from the pool $\mathcal{P}$ that most improve the overall system performance according to one of two proposed model selection algorithms, BC_{greedy} and BC_{search} (see Algorithm 4.3 and 4.4), both of which depend on the incremental Boolean combination module. The pruning module controls the size of the pool by discarding less accurate HMMs that have not been selected for some user-defined time interval. To form and EoHMMs, the incremental Boolean combination module combines the responses from the selected HMMs in the ROC space using all Boolean functions according to the $incrBC$ algorithm (see Algorithm 4.2 in Section 4.3.1). The $incrBC$ algorithm also selects and stores the set ($\mathcal{S}$) of decision thresholds (from each base HMM of the EoHMMs) and Boolean functions that most improves the overall EoHMMs performance. The HMMs that form the EoHMMs, E_k, and the set of decision thresholds and Boolean functions, $\mathcal{S}$, are used during operations.

A set of validation data ($\mathcal{V}$) comprised of *normal* and *anomalous* sub-sequences, is required during the design phase for selection of HMMs, decision thresholds, and Boolean functions. It is managed and input by the system administrator as discussed in Section 4.2.2. When manifestations of novel attacks are discovered, relevant anomalous subsequences are then stored to update the validation set. The HMMs, decision thresholds, and Boolean combinations must then be re-selected to accommodate the new information. Furthermore, the system administrator ensures that the blocks of training data provided for updating the pool of HMMs are clean from intrusions. He is also responsible for selecting and tuning the model management algorithms to trade-off the size of the pool and EoHMMs against overall system accuracy.

The proposed approach inherits all desirable properties of Boolean combination in the ROC space as detailed in the following sections. It covers the whole performance range of

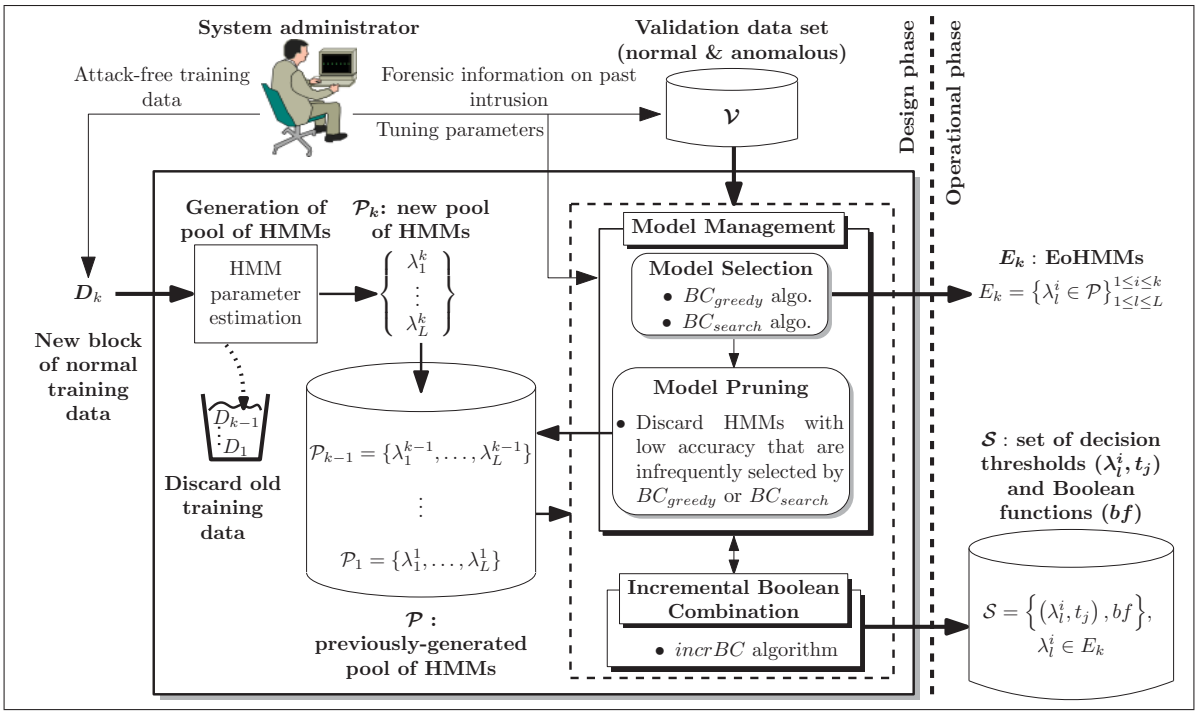


Figure 4.3 Block diagram of the adaptive system proposed for $incrBC$ of HMMs, trained from newly-acquired blocks of data D_k, according to the learn-and-combine approach. It allows for an efficient management of the pool of HMMs and selection of EoHMMs, decision thresholds, and Boolean functions

false and true positive rates, which allows for a flexible selection of the desired operating performance. As conditions change, such as prior probabilities and costs of errors, the overall convex hull of combinations does not change; only the portion of interest. This change shifts the optimal operating point to another vertex on the composite convex hull. The corresponding HMMs are then activated, and their responses are combined according to different decision thresholds and Boolean functions.

4.3.1 Incremental Boolean Combination in the ROC Space

The incremental Boolean combination ($incrBC$) technique summarized in this section has been proposed by the authors (Khreich et al., 2010a,b) to combine the responses from models trained with different number of states and initialization *on a fixed-size data set*. In the learn-and-combine approach for incremental learning, it is adapted to combine the

Algorithm 4.1: Learn-and-combine approach using incremental Boolean combination

input : Training data blocks $D_1, D_2, \ldots, D_K$ that become available over time (normal sub-sequences) Labeled validation data set $\mathcal{V}$ (normal and anomalous sub-sequences)

output: Selected HMMs to form the EoHMMs (E_k) of size $|E_k|$ Selected set ($\mathcal{S}$) of decision thresholds (λ_l^i, t_j) and Boolean functions, each of size 2 to $|E_k|$ detector

select $selection\_algo \in \{BC_{greedy}, BC_{search}\}$; `// choose the model selection algorithm`

set $count_i \leftarrow 0, \forall \lambda_l^i \in \mathcal{P}$; `// counter for unselected HMMs over time`

set LT ; `// lifetime expectancy of models in the pool`

foreach D_k **do**

  train new pool of HMMs, $\mathcal{P}_k = \{\lambda_1^k, \ldots, \lambda_L^k\}$; `// use different no. states and initializations`

  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_k$; `// add the new pool to previously-learned pools`

  switch $selection\_algo$ **do**

    case BC_{greedy}

      $E_k = BC_{greedy}(\mathcal{P}, \mathcal{V})$; `// use greedy selection (Algorithm 4.3)`

    case BC_{search}

      $E_k = BC_{search}(\mathcal{P}, \mathcal{V})$; `// use incremental search (Algorithm 4.4)`

    `// both algorithms call the` $incrBC$ `algorithm (Algorithm 3.1) for selecting and storing the set of best decision thresholds and Boolean functions,` $\mathcal{S} = \{(\lambda_l^i, t_j), bf\}, \lambda_l^i \in E_k$

    $count(\lambda_l^i) \leftarrow count(\lambda_l^i) + 1, \forall \lambda_l^i \in \mathcal{P} \backslash E_k$; `// increment counter for unselected HMMs`

  if $k > LT$ **then**

    prune $\lambda_l^i : count(\lambda_l^i) > LT$; `// discard HMMs that are not selected for` LT `blocks from` $\mathcal{P}$

return E_k and $\mathcal{S}$

responses of HMMs trained with different number of states and initialization, however using new data blocks that are acquired over time.

A *crisp* detector outputs a class label while a *soft* detector assigns scores or probabilities to the input samples, which can be converted to a crisp detector by setting a decision threshold on the scores. Given the responses of a crisp detector on a validation set, the true positive rate (tpr) is the proportion of positives correctly classified over the total number of positive samples. The false positive rate (fpr) is the proportion of negatives

incorrectly classified over the total number of negative samples. A ROC curve is a plot of tpr against fpr. A crisp detector produces a single data point in the ROC plane, while a soft detector produces a ROC curve by varying the decision thresholds. For equal priors and cost of errors, the optimal decision threshold (minimizing overall costs) corresponds to the vertex that is closest to the upper-left corner of the ROC plane. Given two operating points, say a and b, in the ROC space, a is defined as *superior* to b if $fpr_a \leq fpr_b$ and $tpr_a \geq tpr_b$. If one ROC curve has all its points superior to those of another curve, it *dominates* the latter. If a ROC curve has $tpr_x > fpr_x$ for all its points x then, it is a *proper* ROC curve.

In practice, the number of decision thresholds is the number of distinct score values assigned by a soft detector to the validation samples. This number also corresponds to the number of resulting crisp detectors and to the number of vertices on the empirical ROC plot. In fact, an empirical ROC plot is obtained by connecting the observed (tpr, fpr) pairs of a soft detector at each decision threshold. With a finite number of decision thresholds, an empirical ROC plot is a step-like function which approaches a true curve as the number of samples, and hence the number of decision thresholds, approaches infinity. Therefore, it is not necessarily convex and proper. Concavities indicate local performance that is worse than random behavior and may provide diverse information.

The convex hull of an empirical ROC plot (ROCCH) is the smallest convex set containing its vertices, i.e., the piece-wise outer envelope connecting only its superior points. It may be used to combining detectors based on a simple interpolation between their responses (Provost and Fawcett, 2001; Scott et al., 1998). This approach has been called the maximum realizable ROC (MRROC) (Scott et al., 1998) since it represents a system that is equal to, or better than, all the existing systems for all Neyman-Pearson criteria (Neyman and Pearson, 1933). However, the MRROC discards responses from inferior detectors which may provide diverse information for an improved performance. Using Boolean conjunction and disjunction functions to combine the responses of multiple soft detectors in the ROC space have shown and improved performance over the MRROC

(Haker et al., 2005; Tao and Veldhuis, 2008). However, these techniques assume that the detectors are conditionally independent, and their of ROC curves are convex. These assumptions are violated in most real-world applications, especially when detectors are designed using limited and imbalanced training data. Furthermore, the correlation between soft detector decisions also depends on the threshold selections.

The *incrBC* technique applies *all* Boolean functions to combine the responses of multiple crisp or soft detectors, requires no prior assumptions, and selects the thresholds and Boolean functions that improve the MRROC (Khreich et al., 2010b). By applying all Boolean functions to combine the responses of detectors at each corresponding decision threshold, the *incrBC* technique implicitly accounts for the effects of correlation among detectors and accommodates for the concavities in the ROC curves. Indeed, AND and OR rules will not provide improvements for the inferior points that correspond to concavities. Other Boolean functions, for instance those that exploit negations of responses, may however emerge (see Figure 4.4).

The main steps of *incrBC* are presented in Algorithm 3.1. Given a pool of K HMMs $\mathcal{P} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ having n_k, $(k = 1, \ldots, K)$, distinct decision thresholds on a validation set ($\mathcal{V}$), the *incrBC* algorithm starts by combining the responses of the first two HMMs in the pool. Each of the ten Boolean functions is applied to combine the responses of each decision threshold from the first HMM (λ_1, t_i), $i = 1, \ldots, n_1$, with the responses of each decision threshold from the second HMM (λ_2, t_j), $j = 1, \ldots, n_2$. The fused responses are then mapped to points (fpr, tpr) in the ROC space and their ROCCH is computed. Then, *incrBC* selects the emerging vertices which are superior to the ROCCH of original HMMs, stores the set ($\mathcal{S}$) of selected decision thresholds from each HMM and Boolean functions, and updates the ROCCH to include the new emerging vertices. The responses of previously emerging vertices, resulting from the first two HMMs, are then combined with the responses of the third HMM and so on, until the last HMM. The *incrBC* algorithm outputs the final composite ROCCH for visualization and selection of operating points (see Figure 4.4). It also stores the selected set of decision thresholds and Boolean

Algorithm 4.2: $incrBC(\lambda_1, \lambda_2, \ldots, \lambda_K, \mathcal{V})$: Incremental Boolean combination of HMMs in ROC space

input : K HMMs $(\lambda_1, \lambda_2, \ldots, \lambda_K)$ and a validation set $\mathcal{V}$ of size $|\mathcal{V}|$
output: ROCCH of combined HMMs where each vertex is the result of 2 to K combination of crisp detectors. Each combination selects the best decision thresholds from different HMMs (λ_i, t_j) and the best Boolean function, which are stored in the set $(\mathcal{S})$

$n_k \leftarrow$ no. decision thresholds of λ_k using $\mathcal{V}$; `// no. vertices on ROC(`λ_k`) or no. crisp detectors`
$BooleanFunctions \leftarrow \{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$

`// combine responses from the first two HMMs`
compute $ROCCH_1$ of the first two HMMs (λ_1 and λ_2)
allocate $\boldsymbol{F}$ an array of size: $[2, n_1 \times n_2]$; `// temporary storage of combination results`
foreach $bf \in BooleanFunctions$ **do**
  for $i \leftarrow 1$ **to** n_1 **do**
    $\boldsymbol{R_1} \leftarrow (\lambda_1, t_i)$; `// responses of` λ_1 `at decision threshold` t_i `using` $\mathcal{V}$
    for $j \leftarrow 1$ **to** n_2 **do**
      $R_2 \leftarrow (\lambda_2, t_j)$; `// responses of` λ_2 `at decision threshold` t_j `using` $\mathcal{V}$
      $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R_1}, \boldsymbol{R_2})$; `// combine responses using current Boolean function`
      compute (tpr, fpr) of $\boldsymbol{R_c}$ using $\mathcal{V}$; `// map combined responses to ROC space (1 pt)`
      push (tpr, fpr) onto $\boldsymbol{F}$
  compute $ROCCH_2$ *of all ROC points in* $\boldsymbol{F}$
  $n_{ev} \leftarrow$ number of emerging vertices ; `// no. vertices of` $ROCCH_2$ `superior to` $ROCCH_1$
  $\mathcal{S}_2 \leftarrow \{(\lambda_1, t_i), (\lambda_2, t_j), bf\}$ `// set of selected decision thresholds from each HMM and Boolean functions for emerging vertices`

`// combine responses of each successive HMM with the previously-combined responses`
for $k \leftarrow 3$ **to** K **do**
  allocate $\boldsymbol{F}$ of size: $[2, n_k \times n_{ev}]$
  foreach $bf \in BooleanFunctions$ **do**
    for $i \leftarrow 1$ **to** n_{ev} **do**
      $\boldsymbol{R_i} \leftarrow S_{k-1}(i)$; `// responses from previous combinations`
      for $j \leftarrow 1$ **to** n_k **do**
        $\boldsymbol{R_k} \leftarrow (\lambda_k, t_j)$
        $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R_i}, \boldsymbol{R_k})$
        compute (tpr, fpr) of $\boldsymbol{R_c}$ using $\mathcal{V}$
        push (tpr, fpr) onto $\boldsymbol{F}$
  compute $ROCCH_k$ *of all ROC points in* $\boldsymbol{F}$
  $n_{ev} \leftarrow$ number of emerging vertices ; `// no. vertices of` $ROCCH_k$ `superior to` $ROCCH_{k-1}$
  $\mathcal{S}_k \leftarrow \{\mathcal{S}_{k-1}(i), (\lambda_k, t_j), bf\}$ `// set of selected subset from previous combinations, decision thresholds from the newly-selected HMM, and Boolean functions for emerging vertices`
store $\mathcal{S}_k, 2 \leq k \leq K$
return $ROCCH_K$

functions $(\mathcal{S} = \{(\lambda_i, t_j), bf\}, 2 \leq i \leq K; 1 \leq j \leq n_i)$, for each vertex on the composite ROCCH to be applied during operations.

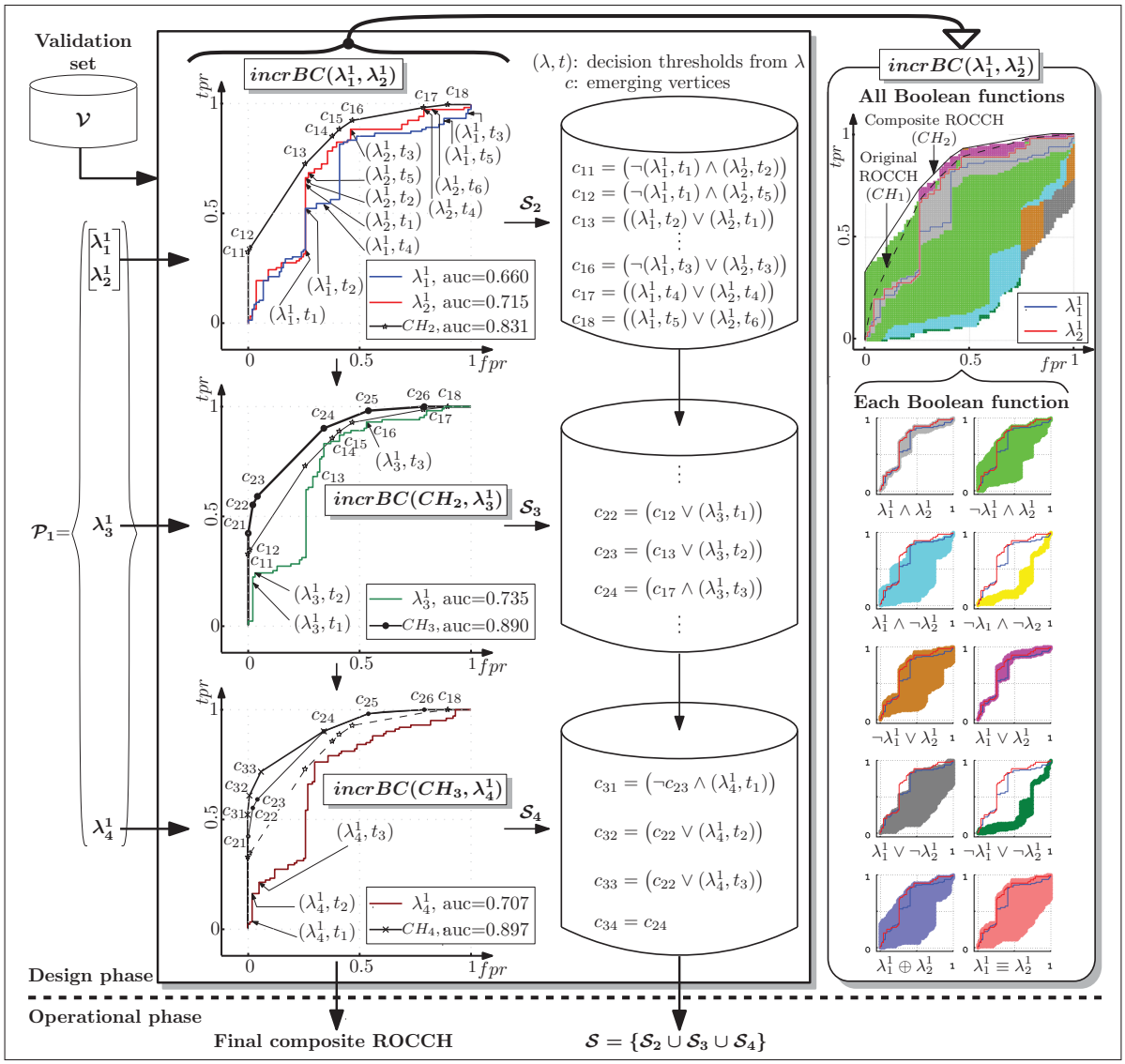


Figure 4.4 An illustration of the steps involved during the design phase of the *incrBC* algorithm employed for incremental combination from a pool of four HMMs $\mathcal{P}_1 = \{\lambda_1^1, \ldots, \lambda_4^1\}$. Each HMM is trained with different number of states and initializations on a block (D_1) of normal data synthetically generated with $\Sigma = 8$ and $CRE = 0.3$ using the BW algorithm (see Section 4.4 for details on data generation and HMM training). At each step, the example illustrates the update of the composite ROCCH (CH) and the selection of the corresponding set ($\mathcal{S}$) of decision thresholds and Boolean functions for overall improved system performance

During the operation phases applied to incremental learning, the system uses one vertex or the interpolation between a pair of vertices on the facets of the final composite ROCCH, depending on the desired false alarm rate. Given a tolerable fpr value, the

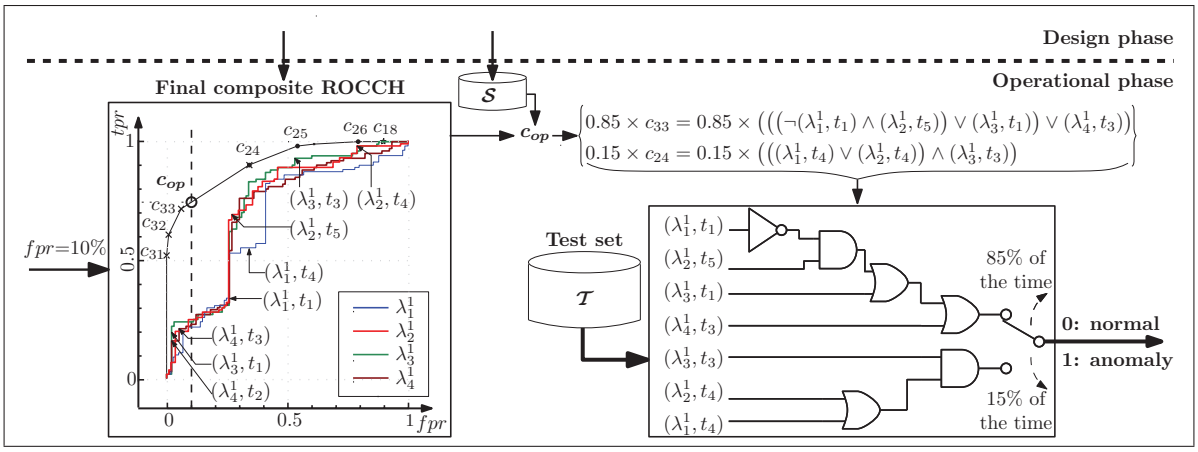


Figure 4.5 An illustration of the *incrBC* algorithm during the operational phase. The example presents the set of decision thresholds and Boolean functions that are activated given, for instance, a maximum specified fpr of 10% for the system designed in Figure 4.4. The operational point (c_{op}) that corresponds to $fpr = 10\%$ is located between the vertices c_{33} and c_{24} of the composite ROCCH, which can be achieved by interpolation of responses (Provost and Fawcett, 2001; Scott et al., 1998). The desired c_{op} is therefore realized by randomly taking the responses from c_{33} with probability value of 0.85 and from c_{24} with probability value of 0.15 . The decision thresholds and Boolean functions of c_{33} and c_{24} are then retrieved from $\mathcal{S}$ and applied for operations

corresponding vertex or pair of vertices (which form an edge intersecting the vertical line at the specified fpr value) are first located on the ROCCH. The decision thresholds and Boolean functions, which have been derived and stored during the design phase, are then extracted and applied during the operational phase (see Figure 4.5). When the operational conditions change, the optimal operating point can be shifted during operations to adapt for the changes by activating a different set of decision thresholds and Boolean functions. If, for instance, c_{32} (in Figure 4.5) has become the optimal operating point because of a different cost of errors or a different tolerance in fpr, then c_{op} can be shifted during operation to $c_{32} = \Big(\Big(\Big(\neg(\lambda_1^1, t_1) \wedge (\lambda_2^1, t_5)\Big) \vee (\lambda_3^1, t_1)\Big) \vee (\lambda_4^1, t_2)\Big)$. The activated decision thresholds and Boolean functions are then updated accordingly.

During the design phase, the worst-case time and memory complexity for *incrBC* of a pool of K HMMs are $\mathcal{O}(Kn_1n_2)$ Boolean operations and $\mathcal{O}(n_1n_2)$ floating point registers, where n_k is the number of distinct decision thresholds of λ_k on a validation set $\mathcal{V}$. They

can be roughly stated as functions of n_1 and n_2, because the number of emerging vertices (n_{ev}) from each successive combination of the remaining HMMs is typically lower than n_1 and n_2. During operations, the computational overhead of the activated set of Boolean functions is lower than that of operating the required number of HMMs. Therefore, the worst-case time and memory complexity is limited to operating the K HMMs.

By selecting crisp detectors from *all* available HMMs in the pool, the *incrBC* algorithm yields to unnecessarily high computational and storage cost when the pool size grows as new blocks of data become available. In addition, HMMs are combined according to the order in which they are stored in the pool. An HMM trained on new data block may capture different underlying data structure and could replace several previously-selected HMMs. Model management strategies are therefore required to manage system resources.

4.3.2 Model Management

When batch learning is performed on a fixed-size data set, a fixed number of classifiers is typically generated. In this case, model management consists in selecting the most accurate and diverse base classifiers from the pool and, if necessary, pruning less relevant classifiers from the pool to reduce computational costs. The selected classifiers form an ensemble to be applied during system operations. Therefore, ensemble selection and pruning are commonly used interchangeably in related literature (Tsoumakas et al., 2009). When learning is performed incrementally, the pool size grows as new blocks of data become available over time. Different ensemble selection and pruning mechanisms are therefore required. Ensemble selection is performed at each learning stage, when a new pool of classifiers is generated from a new block of data, and selected classifiers are deployed for operations. It involves applying some optimization criteria for selection the of the best ensemble of classifiers from all base classifiers in the pool. Pruning involves discarding less relevant members of the pool from future combination according to different criteria. Discarding all unselected classifiers from the pool at each learning stage decreases the system performance as described in Section 4.3.2.2.

4.3.2.1 Model Selection

Ensemble selection techniques are employed to choose a compact ensemble from a large pool of classifiers to increase accuracy and reduce the computational and storage costs of systems deployed during operation. A brute-force search for the optimal ensemble of classifiers results in a combinatorial complexity explosion: the search space comprises 2^K possible ensembles for a pool of K classifiers. Therefore, ensemble selection attempts to select the best ensemble of classifiers from the pool based on different optimization criteria and selection strategies (Tsoumakas et al., 2009; Ulaş et al., 2009). Typical optimization criteria are ensemble accuracy (Ulaş et al., 2009), cross-entropy (Caruana et al., 2004), and ROC-based measures (Rokach et al., 2006). Although there is no universal consensus about the definition and efficiency of diversity measures (Brown et al., 2005), certain approaches have shown promising results (Banfield et al., 2003; Margineantu and Dietterich, 1997; Partalas et al., 2008; Rokach et al., 2006). The selection strategies include ranking-based techniques in which the pool members are ordered according to an evaluation measure on a validation set, and the top classifiers are then selected to form an ensemble (Martinez-Munoz et al., 2009). Search-based ensemble selection is an alternative approach which consists in combining the outputs of classifiers and then selecting the best performing ensemble evaluated on an independent validation set. A heuristic search in the space of all possible ensembles is typically performed, while evaluating the collective merit of selected members (Banfield et al., 2003; Caruana et al., 2004; Margineantu and Dietterich, 1997; Ruta and Gabrys, 2005). Typically, the selection procedure stops when a user-defined maximum number of classifiers is reached (Caruana et al., 2004) or when remaining classifiers provide no further improvement to the ensemble (Ruta and Gabrys, 2005; Ulaş et al., 2009).

Many ensemble selection approaches have been considered for cost-insensitive applications with sufficient amount of training data. Few approaches consider cost-sensitive and limited training data (Fan et al., 2002; Rokach et al., 2006), which are prevalent in anomaly detection applications. Although the performance measures employed in these

cost-sensitive approaches consider the cost of errors and class imbalance, they are either computed at a specific decision threshold or averaged over all decision thresholds. Any change in the operating conditions, such as prior probabilities, cost of errors, and tolerable fpr values, requires reconsidering the combination and selection process.

The proposed ensemble selection algorithms (BC_{greedy} and BC_{search}), described below, employ both rank- and search-based selection strategies to optimize the area under the ROCCH (AUCH). Before applying their selection strategies, they rank all available pool members in decreasing order of AUCH performance on a validation set. In contrast with existing techniques, the proposed selection algorithms exploit the monotonicity of the $incrBC$ algorithm for an early stopping criterion. As show in line 15 and 28 of Algorithm 3.1, $incrBC$ is bound below by the previous ROCCH, and hence guarantees a monotonic improvement in AUCH performance. Furthermore, both algorithms allow for adaptation to changes in prior probabilities and costs of errors by simply shifting the desired operational point, which activates different HMMs, decision thresholds and Boolean functions.

As described in Algorithm 4.3, the BC_{greedy} algorithm employs a greedy search within the pool of HMMs. First, the HMMs in the pool are ranked in decreasing order of their AUCH performance on a validation set and the best HMM is selected. Then, the algorithm starts a cumulative combination of successive HMMs one at the time, selecting only those that improve the AUCH performance, of a cumulative EoHMMs, over a user-defined tolerance value. The algorithms stops when the last HMM in the pool is reached. The worst-case time complexity of this algorithm is $\mathcal{O}(K \log K)$ for sorting, followed by $\mathcal{O}(K)$ for scanning down the ensemble, resulting in $\mathcal{O}(K \log K)$ time complexity w.r.t. the number of Boolean combination of $incrBC$ (see Section 4.3.1).

As described in Algorithm 4.4, the BC_{search} algorithm employs an incremental selective search strategy. It also ranks all HMMs in decreasing order of their AUCH performance on a validation set and selects the HMM with the largest AUCH value. Then, it combines

the selected HMM with each of the remaining HMMs in the pool. The HMM that most improves the AUCH performance of the EoHMMs, is then selected. The cumulative EoHMMs are then combined with each of the remaining HMMs in the pool, and the HMM that provides the largest AUCH improvement to the EoHMMs is selected, and so on. The algorithm stops when the AUCH improvement of the remaining HMMs is lower than a user-defined tolerance value, or when all HMMs in the original ensemble are selected. The worst-case time complexity of this selection algorithm is $O(K^2)$ w.r.t. the number of Boolean operation of $incrBC$ (see Section 4.3.1). However, this is only attained for a zero tolerance value for which the algorithm selects all models with the

Algorithm 4.3: $BC_{greedy}(\mathcal{P},\mathcal{V})$: Boolean combination with a greedy selection

input : Pool of HMMs $\mathcal{P}=\{\lambda_1,\lambda_2,\ldots,\lambda_K\}$ and a validation set $\mathcal{V}$
output: EoHMMs (E) from the pool $\mathcal{P}$

set tol ; `// tolerated improvement in AUCH values`
$j \leftarrow 1$
foreach $\lambda_k \in \mathcal{P}$ **do**
    compute ROC curves and their $ROCCH_k$, using $\mathcal{V}$
sort HMMs $(\lambda_1,\ldots,\lambda_K)$ in descending order of their $AUCH(ROCCH_k)$ values
$\lambda_j = \arg\max_k\{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\}$
$E \leftarrow \lambda_j$; `// select HMM with the largest AUCH value`
$\mathcal{S}_j \leftarrow \lambda_j$;
for $k \leftarrow 2$ **to** K **do**
    $ROCCH_k = incrBC((\mathcal{S}_j,\lambda_k),\mathcal{V})$
    if $AUC(ROCCH_k) \geq AUC(ROCCH_j)+tol$) **then**
        $j \leftarrow j+1$
        $E \leftarrow E \cup \lambda_k$; `// select the` k^{th} `HMM`
        $ROCCH_j \leftarrow ROCCH_k$; `// update the convex hull`
        if $k=2$ **then**
            $\mathcal{S}_j \leftarrow \{(\lambda_j,t_i),(\lambda_k,t_{i'}),bf\}$
            `// set of selected decision thresholds from each HMM and Boolean functions; derived from` $incrBC$ `for emerging vertices`
        else
            $\mathcal{S}_j \leftarrow \{\mathcal{S}_{j-1}(i),(\lambda_k,t_{i'}),bf\}$
            `// set of selected subset from previous combinations, decision thresholds from the newly-selected HMM, and Boolean functions; derived from` $incrBC$ `for emerging vertices`
store $\mathcal{S}_j,\ 2 \leq j \leq K$
return E

optimum order for combination. In practice, however, depending on the value of the tolerance, the selected number of models $k < K$ and also the computational time, can be traded-off as desired.

Algorithm 4.4: $BC_{search}(\mathcal{P}, \mathcal{V})$: Boolean combination with an incremental selective search

input : Pool of HMMs $\mathcal{P} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ and a validation set $\mathcal{V}$
output: EoHMMs (E) from the pool $\mathcal{P}$

set tol ; // tolerated improvement in AUCH values
foreach $\lambda_k \in \mathcal{P}$ **do**
  compute ROC curves and their $ROCCH_k$, using $\mathcal{V}$
sort HMMs $(\lambda_1, \ldots, \lambda_K)$ in descending order of their $AUCH(ROCCH_k)$ values
$\lambda_1 = \arg\max_k \{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\}$
$E \leftarrow \lambda_1$; // select HMM with the largest AUCH value
foreach $\lambda_k \in \mathcal{P} \backslash E$ **do** // remaining HMMs in $\mathcal{P}$
  $ROCCH_k$=$incrBC$ $((\lambda_1, \lambda_k), \mathcal{V})$
$\lambda_2 = \arg\max_k \{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P} \backslash E\}$
$E \leftarrow E \cup \lambda_2$; // select HMM that provides the largest AUCH improvement to E
$ROCCH_1 \leftarrow ROCCH_2$; // update the convex hull
$\mathcal{S}_2 \leftarrow \{(\lambda_1, t_i), (\lambda_2, t_{i'}), bf\}$ // Set of selected decision thresholds from each HMM and Boolean functions; derived from $incrBC$ for emerging vertices
$j \leftarrow 3$
repeat
  foreach $\lambda_k \in \mathcal{P} \backslash E$ **do**
    $ROCCH_k$=$incrBC$ $((\mathcal{S}_{j-1}, \lambda_k), \mathcal{V})$
  $\lambda_j = \arg\max_k \{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P} \backslash E\}$
  $E \leftarrow E \cup \lambda_j$
  $ROCCH_{j-1} \leftarrow ROCCH_j$
  $\mathcal{S}_j \leftarrow \{\mathcal{S}_{j-1}(i), (\lambda_j, t_{i'}), bf\}$ // Set of selected subset from previous combinations, decision thresholds from the newly-selected HMM, and Boolean functions; derived from $incrBC$ for emerging vertices
  $j \leftarrow j + 1$
until $AUCH(ROCCH_j) \leq AUCH(ROCCH_{j-1}) + tol$; // no further improvement
store $\mathcal{S}_j, 2 \leq j \leq K$
return E

For both ensemble selection algorithms the AUCH improvement is the only user-defined parameters. If desired, it is also possible to impose a maximum number of HMMs as a stopping criterion. The performance measure employed to guide the search is not restricted to AUCH. Other measures such as the partial AUCH or the true positive rate at a required false positive rate, can be also employed for a region-specific performance.

Although these performance measures summarize the ROC space with a single value to guide the search for potential ensemble members, the final composite ROCCH is always stored for visualization of the whole range of performance and adaptation of the operating point to environmental changes.

Figure 4.6 presents an example illustrating the level of performance achieved by $incrBC$, BC_{greedy}, and BC_{search} algorithms. In Figure 4.6, all algorithms achieve a comparable ROCCH and AUC performance. However, as shown in the legends, the size of the EoHMMs obtained by BC_{search} is four HMMs compared to seven HMMs selected by BC_{greedy} from the pool of eight HMMs, which are all combined by $incrBC$. For comparison, the selected set of decision thresholds and Boolean functions for the vertices denoted by C_1 and C_2 on the ROCCHs of Figure 4.6 are shown below for each algorithms:

$$incrBC \begin{cases} C_1 = \big(\big(\big(\big(\big(\big(\big(\big((\lambda_1^1,t_1) \vee (\lambda_2^1,t_1)\big) \wedge \neg(\lambda_3^1,t_1)\big) \vee (\lambda_4^1,t_1)\big) \vee (\lambda_1^2,t_2)\big) \vee (\lambda_2^2,t_2)\big) \wedge (\lambda_3^2,t_2)\big) \wedge (\lambda_4^2,t_2)\big) \\ C_2 = \big(\big(\big(\big(\big(\big(\big(\big((\lambda_1^1,t_1) \vee (\lambda_2^1,t_1)\big) \wedge \neg(\lambda_3^1,t_1)\big) \vee (\lambda_4^1,t_1)\big) \vee (\lambda_1^2,t_2)\big) \vee (\lambda_2^2,t_2)\big) \wedge (\lambda_3^2,t_2)\big) \wedge (\lambda_4^2,t_2)\big) \end{cases}$$

$$BC_{greedy} \begin{cases} C_1 = \big(\big(\big(\big(\big(\big((\neg(\lambda_3^2,t_1) \wedge (\lambda_4^2,t_1)) \vee (\lambda_3^1,t_1)\big) \vee \neg(\lambda_2^2,t_1)\big) \wedge (\lambda_2^1,t_1)\big) \wedge \neg(\lambda_1^2,t_1)\big) \vee (\lambda_1^1,t_1)\big) \\ C_2 = \big(\big(\big(\big(\big(\big((\neg(\lambda_3^2,t_1) \wedge (\lambda_4^2,t_1)) \vee (\lambda_3^1,t_1)\big) \vee \neg(\lambda_2^2,t_1)\big) \wedge (\lambda_2^1,t_1)\big) \vee (\lambda_1^2,t_2)\big) \vee (\lambda_1^1,t_2)\big) \end{cases}$$

$$BC_{search} \begin{cases} C_1 = \big(\big((\neg(\lambda_3^2,t_1) \wedge (\lambda_2^2,t_1)) \vee (\lambda_3^1,t_1)\big) \vee (\lambda_1^2,t_1)\big) \\ C_2 = \big(\big((\neg(\lambda_3^2,t_1) \wedge (\lambda_2^2,t_1)) \vee (\lambda_3^1,t_2)\big) \vee (\lambda_1^2,t_2)\big) \end{cases}$$

This example demonstrates the efficiency of the proposed ensemble selection techniques in forming compact EoHMMs by exploiting the new information provided from newly-acquired data while maintaining a high level of performance. More extensive simulations and detailed discussions are presented in Section 4.5.

4.3.2.2 Model Pruning

Pruning less relevant models is essential to restrict the pool size with incremental learning from growing indefinitely as new blocks of data become available. As described in the previous subsection, BC_{greedy} and BC_{search} algorithms are designed to form the most compact EoHMMs from a pool $\mathcal{P}$ of HMMs. According to the learn-and-combine approach new pools of HMMs are accumulated from successive blocks of data. When the

number of data blocks increases over time, an increasingly large storage space is required

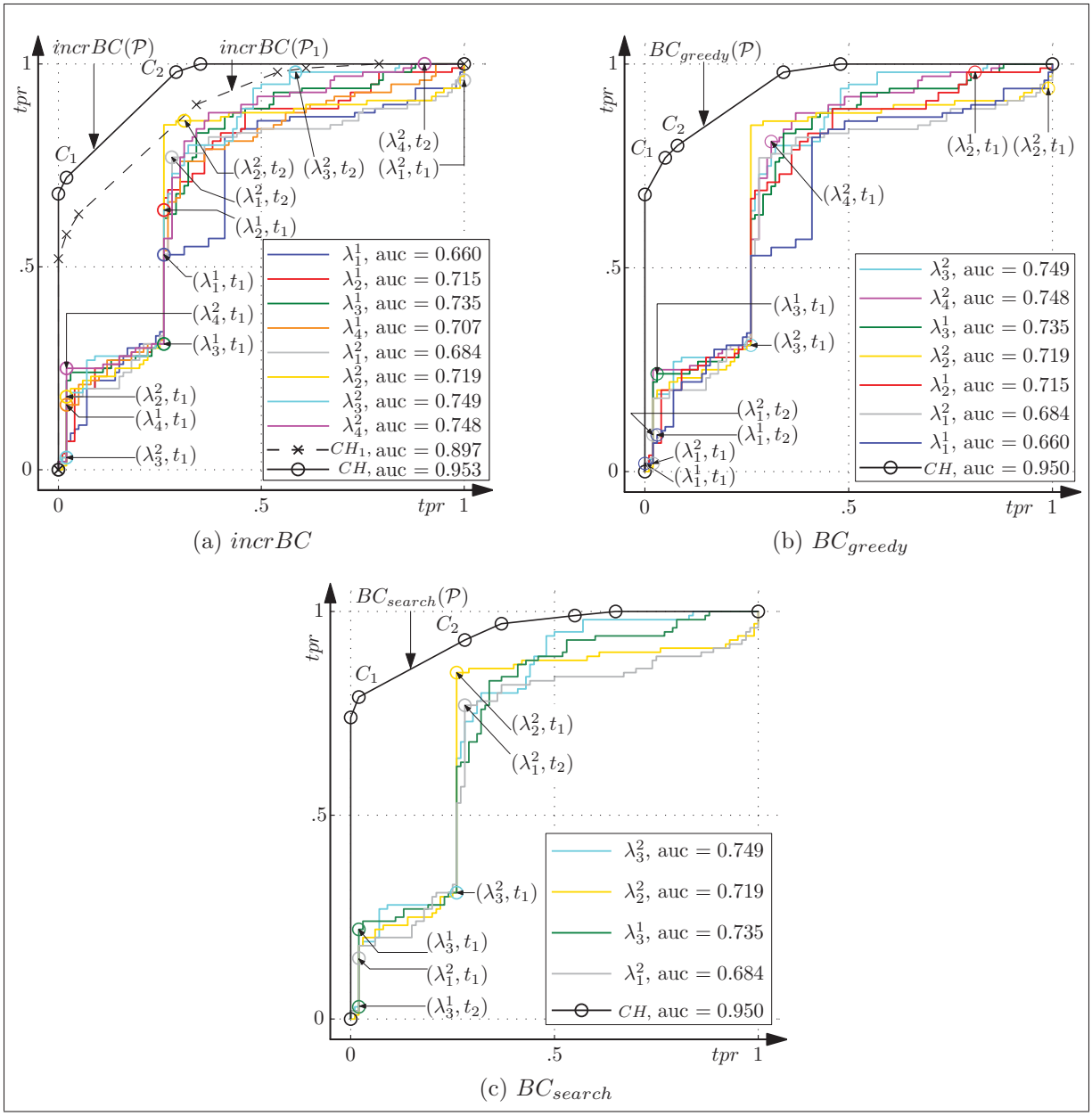


Figure 4.6 A comparison of the composite ROCCH (CH) and AUC performance achieved with the HMMs selected according to the BC_{greedy} and BC_{search} algorithms. Suppose that a new pool of four HMMs $\mathcal{P}_2 = \{\lambda_1^2, \ldots, \lambda_4^2\}$ is generated from a new block of training data (D_2), and appended to the previously-generated pool, $\mathcal{P}_1 = \{\lambda_1^1, \ldots, \lambda_4^1\}$, in the example presented in Section 4.3.1 (Figure 4.4). The *incrBC* algorithm combines all available HMMs in $\mathcal{P} = \{\lambda_1^1, \ldots, \lambda_4^1, \lambda_1^2, \ldots, \lambda_4^2\}$, according to their order of generation and storage, while BC_{greedy} and BC_{search} start by ranking the members of $\mathcal{P}$ according to AUC values and then apply their ensemble selection strategies

for storing these HMMs. In addition, the time complexity of BC_{greedy} and BC_{search} algorithms also increases over time. Discarding unselected classifiers immediately from the pool yields to knowledge corruption and hence to a decline system performance. Although these classifiers did not provide any improvement to the cumulative EoHMMs, they may provide diverse information to the newly-generated pool of HMMs, which have different view of the data, to increase system performance. One solution to this issue is to discard repeatedly unselected HMMs over time. A counter is therefore assigned to each HMM in the pool indicating the number of blocks for which an HMM was not selected as an ensemble member (see Algorithm 4.1). An HMM is then pruned from the pool, according to a user-defined life time (LT) expectancy value of unselected models. For instance, with an $LT = 3$ all HMMs that have not been selected after receiving three blocks of data, as indicated by their counters, are discarded from the pool.

4.4 Experimental Methodology

4.4.1 Data Sets

The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets[3]. The UNM data sets are commonly used for benchmarking anomaly detections based on system calls sequences (Warrender et al., 1999). In related work, intrusive sequences are usually labeled by comparing normal sequences, using the Sequence Time-Delay Embedding (STIDE) matching technique. This labeling process considers STIDE responses as the ground truth, and leads to a biased evaluation and comparison of techniques, which depends on both training data size and detector window size. To confirm the results on system calls data from real processes, the same labeling strategy is used in this work. However fewer sequences are used to train the HMMs to alleviate the bias. Therefore, STIDE is first trained on all the available normal data, and then used to label the corresponding sub-sequences from the ten sequences available for testing. The resulting labeled sub-sequences are concatenated,

[3] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

then divided into blocks of equal sizes, one for validation and the other for testing. During the experiments, smaller blocks of normal data (100 to 1,000 sub-sequences) are used for training the HMMs as normal system call observations are very redundant. In spite of labeling issues, redundant training data, and unrepresentative test data, UNM sendmail data set is the mostly used in literature due to limited publicly available system call data sets.

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training and labeling) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations. The data generator is based on the Conditional Relative Entropy (CRE) of a source; it is closely related to the work of Tan and Maxion (Tan and Maxion, 2003). The CRE is defined as the conditional entropy divided by the maximum entropy ($MaxEnt$) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE is given by $CRE = \frac{-\sum_x p(x) \sum_y p(y|x) \log p(y|x)}{MaxEnt}$, where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a sequence of system calls, the conditional probability, $p(y \mid x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $M = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly sequence

that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly sequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly sequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test[4].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE = 1$) is generated, using the same alphabet size Σ, and segmented into sub-sequences of a desired length using a sliding window with a fixed size of AS. Then, the original generative Markov chain is used to compute the likelihood of each sub-sequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol transitions in the sequence of size AS. This ensures that the anomalous sequences of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into this sequence at random according to a mixing ratio.

4.4.2 Experimental Protocol

The experiments conducted in this chapter using the data generator simulate a simple process, with $\Sigma = 8$ and $CRE = 0.3$ and a more complex process, with $\Sigma = 50$ and

[4]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occur in high frequency over a short period of time.

$CRE = 0.4$. The sizes of injected anomalies are assumed equal to the detector window sizes $AS = DW = 4$. For both scenarios, the presented results are for validation and test sets that comprise 75% of normal and 25% of anomalous data. Although not show in this chapter, various experiments have been conducted using different values of AS and DW, and different ratios of normal to anomalous data. These experiments produced similar results and hence the discussion presented in the next section hold.

Figure 4.7 illustrates the steps involved for estimating HMM parameters. Given the first block of training data D_1, different discrete-time ergodic HMMs are trained with various number of hidden states $N = [N_{min}, \ldots, N_{max}]$ using the 10-fold cross validation (10-FCV). The training block D_1, which only comprises normal sub-sequences, is randomly partitioned into $K = 10$ sub-blocks of equal size. For each fold k, each of the learning techniques described below is used to estimate HMM parameters using $K-1$ sub-blocks. The Forward algorithm is then used to evaluate the log-likelihood on the remaining sub-block, which is used as a stopping criterion to reduce the overfitting effects. The training process is repeated ten times using a different random initialization to avoid local maxima, which provides 100 ROC curves for each N value. Finally, the model that gives the highest area under its convex hull on a validation set ($\mathcal{V}$) comprising normal and anomalous sub-sequences is selected, which results an HMM for each N value.

Figure 4.8 presents HMM parameter estimation according to each learning technique from successive blocks of data for each N value. When the second training block D_2 becomes available, the batch Baum-Welch (BBW) algorithm discards the previously learned HMMs and restarts the training procedure with all cumulative data ($D_1 \cup D_2$), while the Baum-Welch (BW) algorithm restarts the training using D_2 only providing HMMs for incremental combination according to the *incrBC* algorithm. The on-line BW (OBW) and incremental BW (IBW) algorithms resume the training from the previously-learned HMMs (λ_N^1) using only the current block (D_2). The OBW algorithm re-estimates HMM parameters based on each sub-sequence without iterations, while the IBW algorithm re-iterates on D_2 and integrates the new information with previously-learned model at each

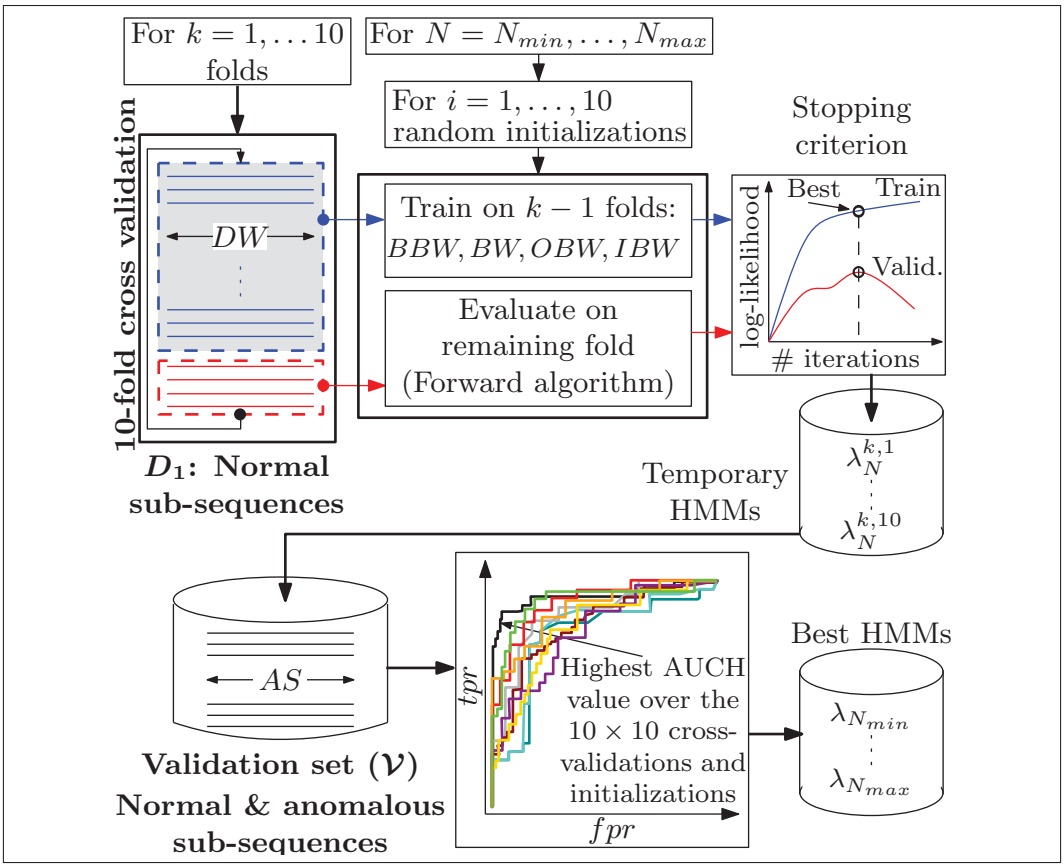


Figure 4.7 Overall steps involved to estimate HMM parameters and select HMMs with the highest AUCH for each number of states from the first block (D_1) of normal data, using 10-FCV and ten random initializations

iteration, until the stopping criteria are met (see Appendix III and Khreich et al., 2009a).

The area under the ROC curve (AUC), has been largely suggested as a robust scalar summary of classifiers performance (Huang and Ling, 2005; Provost and Fawcett, 2001). The AUC assesses ranking in terms of class separation – it evaluates how well a classifier is able to sort its predictions according to the confidence they are assigned. For instance, with an $AUC = 1$ all positives are ranked higher than negatives indicating a perfect discrimination between classes. A random classifier has an $AUC = 0.5$ that is both classes are ranked at random. If the AUC or the partial AUC (Walter, 2005) are not significantly different, the shape of the curves might need to be looked at. It may also be useful to observe the tpr for a fixed fpr of particular interest. Since the MRROC

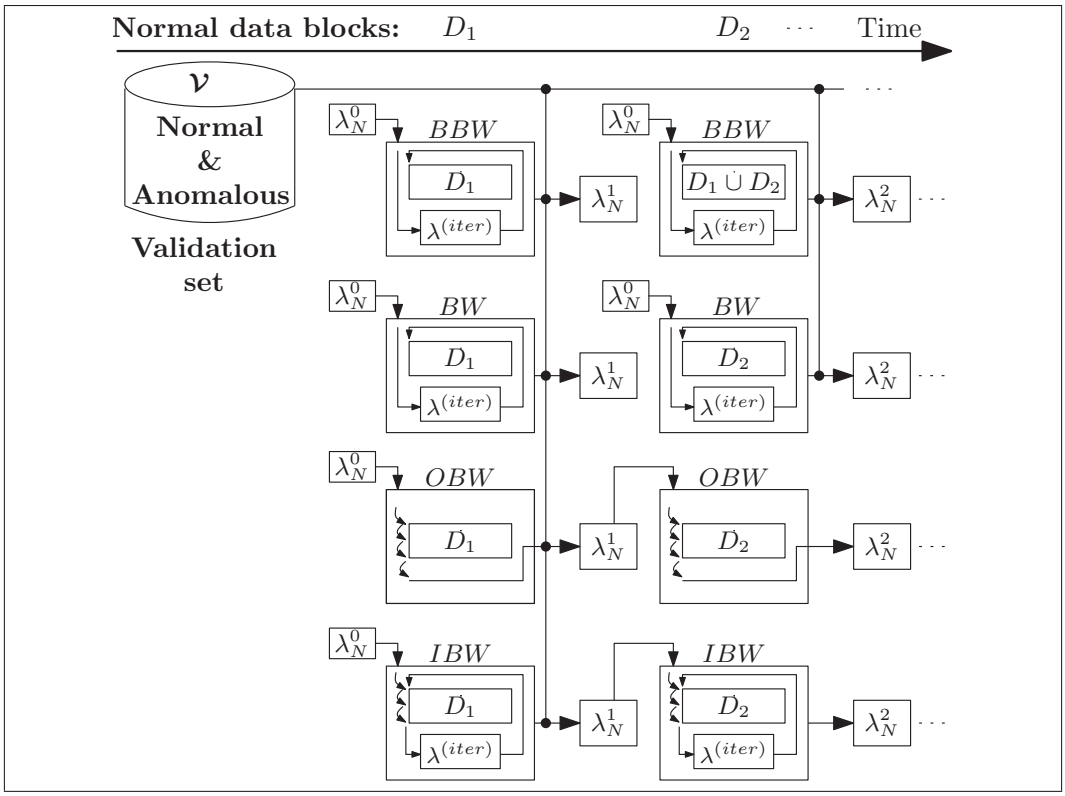


Figure 4.8 An illustration of HMM parameter estimation according to each learning technique (BBW, BW, OBW, and IBW) when subsequent blocks of observation sub-sequences $(D_1, D_2, D_3, \ldots)$ become available

can be applied to any ROC curve, the performance in all experiments are measured with reference to the ROCCH, including the area under the convex hull (AUCH) and the tpr at $fpr = 0.1$. When working with the synthetic data, the whole procedure is replicated ten times with different training, validation and testing sets, and the median results are presented along with the lower and upper quartiles to provide statistical confidence intervals.

4.5 Simulation Results

The first subsection presents the performance of the proposed learn-and-combine approach for incremental learning of new data without employing a model management strategy. In this case, a pool of HMMs for a new block of training data is combined with all previously-generated HMMs according to the $incrBC$ algorithm. The second

subsection shows the impact on performance of the ensemble selection algorithms and of the pruning strategies for limiting the pool size.

4.5.1 Evaluation of the Learn-and-Combine Approach:

A. HMMs Trained with a Fixed Number of States on Successive Blocks of Data.

The first experiment involves ergodic HMMs trained with $N = 6$ states on ten blocks of data. The training, validation and testing data are generated synthetically as described in Section 4.4.1, with $\Sigma = 8$ and $CRE = 0.3$. Each training block D_k $(k = 1, \ldots, 10)$ comprises 50 normal sub-sequences, each of size $DW = 4$. Each validation ($\mathcal{V}$) and test ($\mathcal{T}$) set comprises 200 sub-sequences, each of size $AS = 4$. In both data sets, the ratio of normal to anomalous sub-sequences is $4 : 1$. The training and validation of HMMs follow the methodology described in Section 4.4.2. For each block D_k, a pool $\mathcal{P}_k$ is obtained by applying the BW algorithm to D_k using 10-FCV and ten different random initializations, and selecting the HMM ($\lambda^k_{N=6}$) that gives the highest AUCH on $\mathcal{V}$ (see Figure 4.7). $\mathcal{P}_k$ is then appended to a pool $\mathcal{P}$ ($\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_k$). After receiving the last block of data, the pool $\mathcal{P} = \left\{\lambda^1_{N=6}, \ldots, \lambda^{10}_{N=6}\right\}$ of size $|\mathcal{P}| = 10$ HMMs is provided for incremental combination according to the *incrBC* algorithm. The reference BBW follows the same training procedure but with cumulative blocks of data, and both OBW and IBW algorithms resume the training from the previously-learned HMMs using only the current block of data (see Figure 4.8). Figure 4.9 compares the median AUCH performance (Figure 4.9a) and the median *tpr* values at $fpr = 0.1$ (Figure 4.9b) as well as their lower and upper quartiles over ten replications for each learning technique. The performance obtained with BBW, OBW and IBW algorithms are compared to that of the *incrBC* algorithm combining the responses of the HMMs in $\mathcal{P}$, incrementally, over the ten blocks of data. The performance achieved by combining the outputs of the HMMs in $\mathcal{P}$, over the ten blocks of data, with the static median (MED) and majority vote (VOTE) fusion functions are also provided for reference.

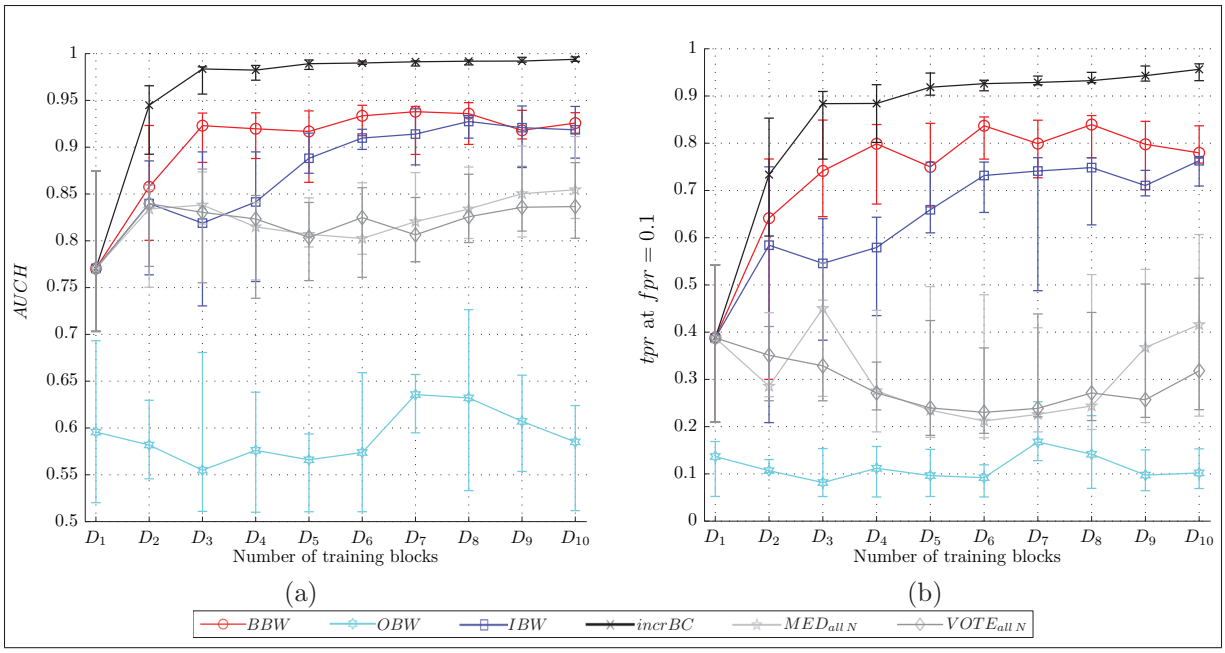


Figure 4.9 Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. The HMMs are trained according to each technique with $N = 6$ states for each block of data providing a pool of size $|\mathcal{P}| = 1, 2, \ldots, 10$ HMMs. Error bars are lower and upper quartiles over ten replications

As shown in Figure 4.9, the learn-and-combine approach using the *incrBC* algorithm provides the highest level of performance (with the lowest variances) among all incremental learning techniques over the whole range of results. Performance is significantly higher than that of the reference BBW, especially when provided with limited training data, as shown in the performance achieved with the first few blocks. The level of performance provided by the static MED and VOTE combiners is lower (with higher variances) versus other techniques, and oscillates around their initial values. Not surprisingly, the OBW algorithm has achieved the worst level of performance as one pass over a limited data is insufficient to capture its underlying structure. In contrast, the IBW algorithm has provided a higher level of performance than that of OBW as it iterates over each block and employs a fixed learning rate to integrate the newly-acquired information into HMM parameters at each iteration (see Appendix III).

B. HMMs Trained with Different Number of States on Successive Blocks of Data.

In order to investigate the effects of the number of states on the performance of each technique, the previous experiment is conducted with a number of states ranging from $N = 4$ to 12. For each N value, a pool $\mathcal{P}_N$ is obtained by applying the BW algorithm to each block D_k using 10-FCV and ten different random initializations, and selecting the HMM (λ_N^k) that gives the highest AUCH on $\mathcal{V}$ (see Figure 4.7 and 4.8). After receiving the last block of data, nine pools are generated, a pool $\mathcal{P}_N = \left\{\lambda_N^1, \ldots, \lambda_N^{10}\right\}$ for each state value $N = 4, \ldots, 12$, each of size $|\mathcal{P}_N| = 10$ HMMs. The responses of HMMs in each pool $\mathcal{P}_N$ are incrementally combined using the $incrBC$ algorithm over each block. The number of states that achieved the highest average level of performance on each block of data is selected. The same procedure is also applied for BBW, OBW and IBW algorithms (see Figure 4.7 and 4.8). In addition, the learn-and-combine approach is employed to incrementally combine the HMMs trained using the whole range of states over the ten blocks. The nine pools are therefore concatenated into one pool $\mathcal{P} = \left\{\lambda_{N=4}^1, \ldots \lambda_{N=12}^1; \ldots; \lambda_{N=4}^{10}, \ldots \lambda_{N=12}^{10}\right\}$ of size $|\mathcal{P}| = 90$ HMMs, and incrementally combined according to the $incrBC$ algorithm over all the successive blocks ($incrBC_{all\,N}$). The HMMs in $\mathcal{P}$ are also combined according to the median ($MED_{all\,N}$) and majority vote ($VOTE_{all\,N}$) functions for comparison. Figure 4.10 presents the median results of the experiments as well as their lower and upper quartiles over ten replications for each learning technique. For BBW, OBW, IBW, and $incrBC$ technique, the number of states that achieved the highest average level of performance on each block of data is indicated with each median value.

As shown in Figure 4.10, the best number of states varies from one block to the next and different among all techniques. Therefore, combination of HMMs each trained with different number of states and random initializations may increase the ensemble diversity and improve system performance. This is clearly seen in the performance achieved with $incrBC_{all\,N}$, which is significantly higher than all other techniques. The previous obser-

vations hold true for the remaining techniques. In fact, to obtain the number of states which provides the best performance for each learning techniques on each data block, the HMMs are trained and evaluated with each N values according to different random initializations. Instead of selecting the single "best" HMM from the pool, $incrBC_{all\,N}$ combines all HMMs with a minimal overhead. As expected, the level of performance achieved by the learning algorithms increases when provided with more training blocks. Although $incrBC$ and $incrBC_{all\,N}$ algorithms are still capable of achieving the highest level of performance, the increased performance over other techniques is relatively lower, as the combined HMMs become more positively correlated.

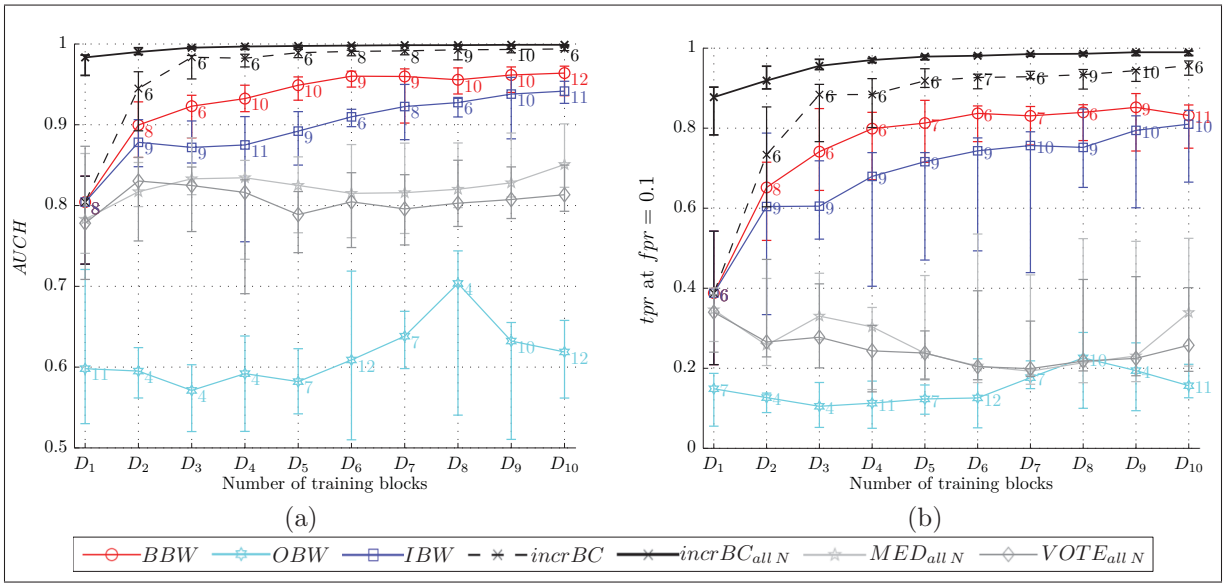


Figure 4.10 Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. The HMMs are trained according to each technique with nine different states ($N = 4, 5, \ldots, 12$) for each block of data providing a pool of size $|\mathcal{P}| = 9, 18, \ldots, 90$ HMMs. Numbers above points are the state values that achieved the highest average level of performance on each block. Error bars are lower and upper quartiles over ten replications

The previous results are also confirmed on the more complex synthetic data in Figure 4.11 and on sendmail data in Figure 4.12. The training and validation of the ergodic HMMs with 20 different number of states ($N = 5, 10, \ldots, 100$), is carried out according to the methodology described in Section 4.4. For each of the ten data blocks, 20 HMMs are

generated and appended to the pool, which gives a pool of size $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs. For each replication of the synthetic data, the training is conducted on ten successive blocks each comprising 500 sub-sequences of length $DW = 4$ symbols, the validation set $\mathcal{V}$ comprises $2,000$ sub-sequences and the test set $\mathcal{T}$ comprises $5,000$ sub-sequences. For sendmail data, the training is conducted on ten successive blocks each comprising 100 sub-sequences of length $DW = 4$ symbols, each $\mathcal{V}$ and $\mathcal{T}$ comprise 450 sub-sequences. In both cases, the anomaly size is $AS = 4$ symbols and the ratio of normal to anomalous sequences is $4 : 1$. Again, the incremental learn-and-combine approach provides a significantly higher level of performance than the BBW, OBW, IBW, MED and VOTE techniques.

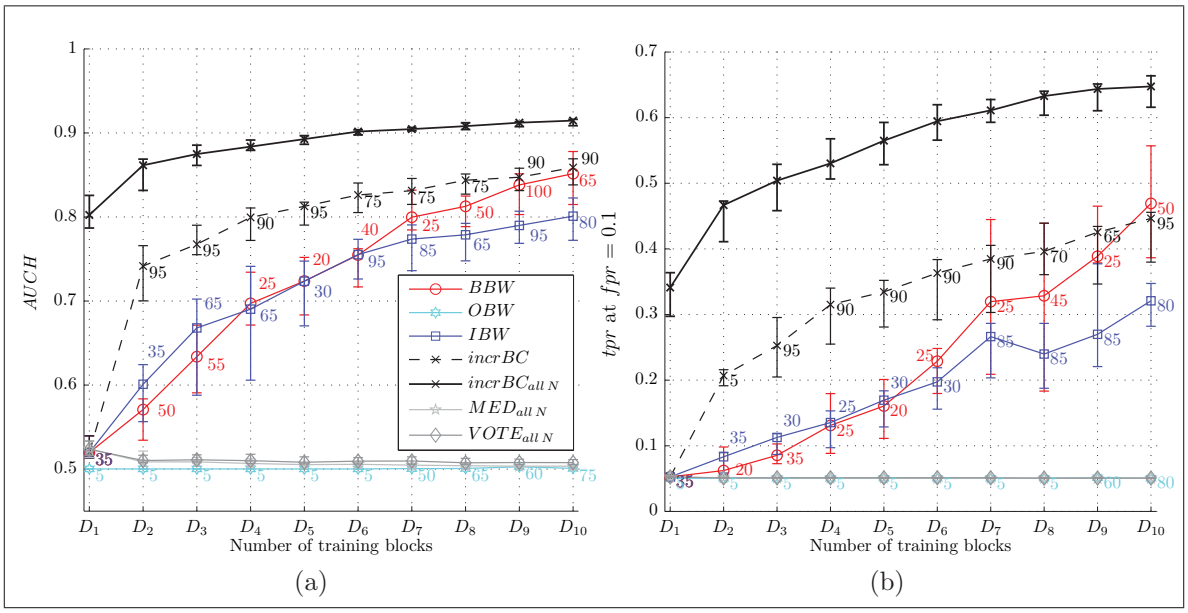


Figure 4.11 Results for synthetically generated data with $\Sigma = 50$ and $CRE = 0.4$. The HMMs are trained according to each technique with 20 different states $(N = 5, 10, \ldots, 100)$ for each block of data providing a pool of size $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs. Numbers above points are the state values that achieved the highest average level of performance on each block. Error bars are lower and upper quartiles over ten replications

The level of performance achieved by applying the learn-and-combine approach to HMMs trained on each newly-acquired block of data always provide the highest overall accuracy. In particular, the results have shown that it provides a higher level of performance than

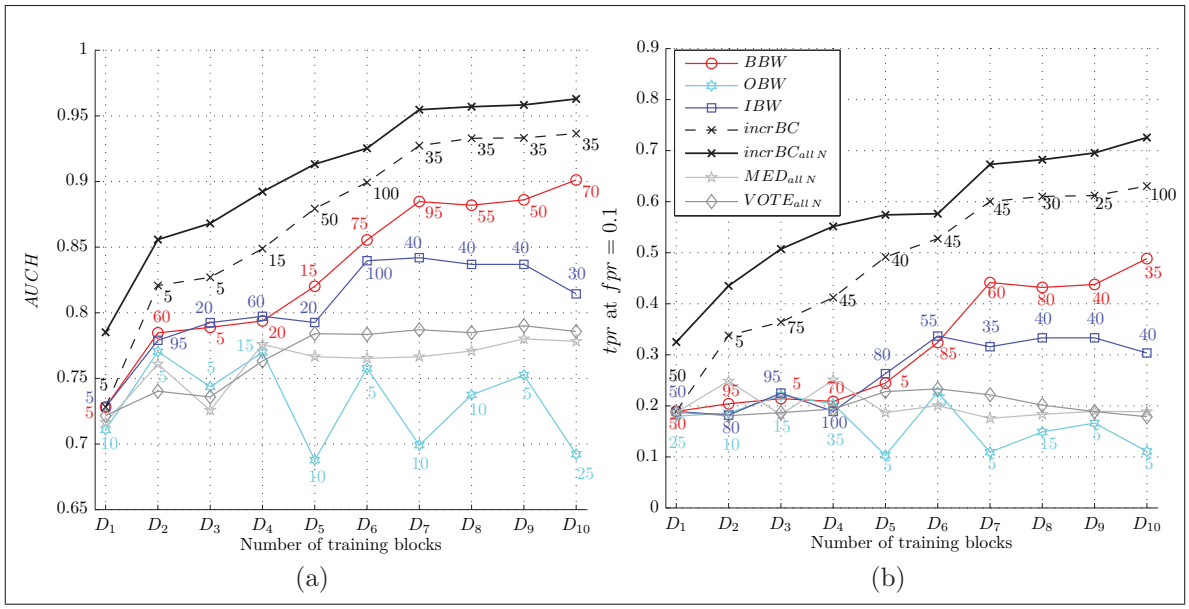


Figure 4.12 Results for sendmail data. The HMMs are trained according to each technique with 20 different states ($N = 5, 10, \ldots, 100$) for each block of data providing a pool of size $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs. Numbers above points are the state values that achieved the highest level of performance on each block

the reference BBW, especially when provided with limited data. This stems from the capabilities of the *incrBC* algorithm in effectively exploiting the diverse and complementary information provided from the pool of HMMs trained with different number of states and different initializations, and from the newly-acquired data. In fact, BW training optimizes HMM parameters locally, which results in converging to different local maxima. Furthermore, HMMs trained with a different number of states allow for capturing different structures of the underlying data. In addition, the newly-acquired blocks of training data provide different views of the true underlying distribution. According to the bias-variance decomposition (Breiman, 1996; Domingos, 2000), segmenting the training data introduces bias and decreases the generalization ability of the individual classifiers. However, this increases the variances among classifiers trained on each data block. An increased generalization ability is therefore achieved by combining the outputs of those classifiers and hence improved level of performance.

The MED and VOTE fusion functions have shown incapable of increasing the level of performance compared to the *incrBC* technique. This reflects their inabilities to effectively exploit the information provided from the validation set. The MED function directly combines HMMs likelihood values for each sub-sequence in the test, while VOTE considers the crisp decisions from HMMs at optimal operating thresholds or equivalently at equal error rates. In contrast, the *incrBC* algorithm applies ten Boolean functions to the crisp decisions provided by each threshold from the first HMM to those provided by the second HMM, and then selects the decision thresholds and Boolean functions that improve the overall ROCCH of the validation set $\mathcal{V}$. Shifting the decision thresholds for each HMM in the ensemble before combining their responses has shown to largely increase the diversity of the ensemble. The *incrBC* algorithm is effectively designed to take advantage from these threshold-based complementary information and to select those that most improve the level of performance.

4.5.2 Evaluation of Model Management Strategies:

A. Model Selection.

In the previously conducted experiments, using the synthetic ($\Sigma = 50$ and $CRE = 0.4$) and sendmail data sets, the pool $\mathcal{P}$ comprised 20 HMMs for each data block (an HMM for each number of states $N = 5, 10, \ldots, 100$), yielding a pool of size $|\mathcal{P}| = 200$ HMMs after receiving the ten blocks of data. This is also the size of the EoHMMs ($|E| = 20, 40, \ldots, 200$ HMMs), since for each block D_k, all available HMMs in $\mathcal{P}$ were selected and incrementally combined according to the learn-and-combine approach ($incrBC_{all\,N}$). Figure 4.13 presents the results of the combined EoHMMs previously considered using the synthetically generated data with $\Sigma = 50$ and $CRE = 0.4$. These results belong to the first replication of Figure 4.11. AUCH performance of the $incrBC_{all\,N}$ that combines all available HMMs from the pool is compared to that of the proposed ensemble selection algorithms, BC_{greedy} (Algorithm 4.3) and BC_{search} (Algorithm 4.4). For each block, the size of the EoHMMs selected according to each technique is shown on the respective

curves of the figure. The performance of the BBW, with the selected (best) state value at each block, is also shown for reference.

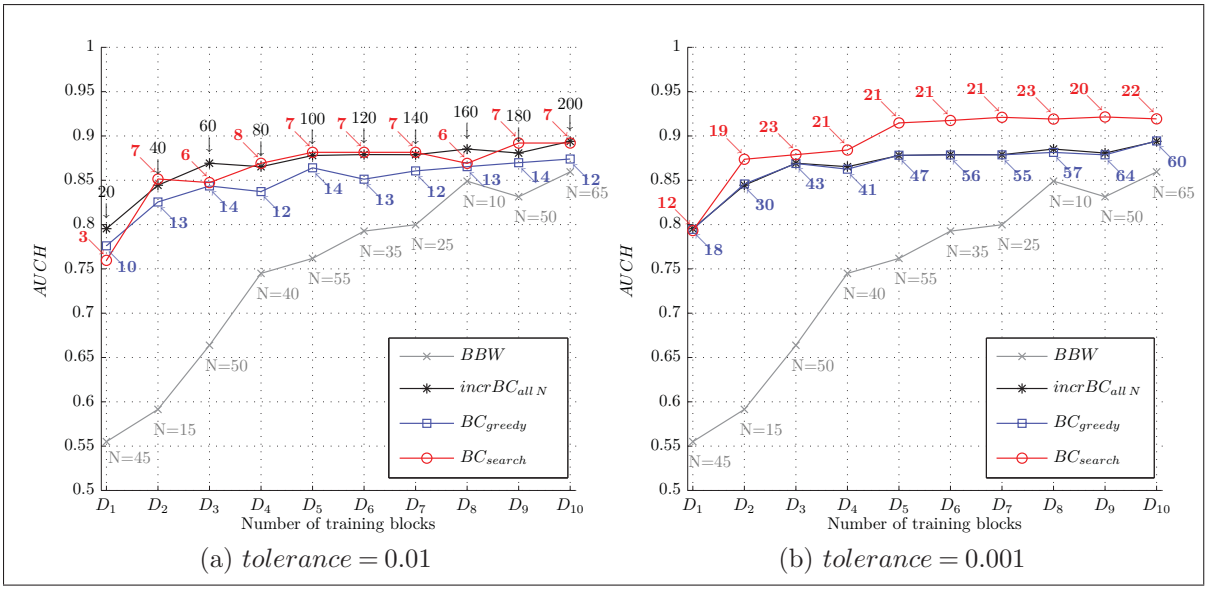


Figure 4.13 Ensemble selection results for synthetically generated data with $\Sigma = 50$ and $CRE = 0.4$. These results are for the first replication of Figure 4.11. For each block, the values on the arrows represent the size of the EoHMMs ($|E|$) selected by each technique from the pool of size $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs

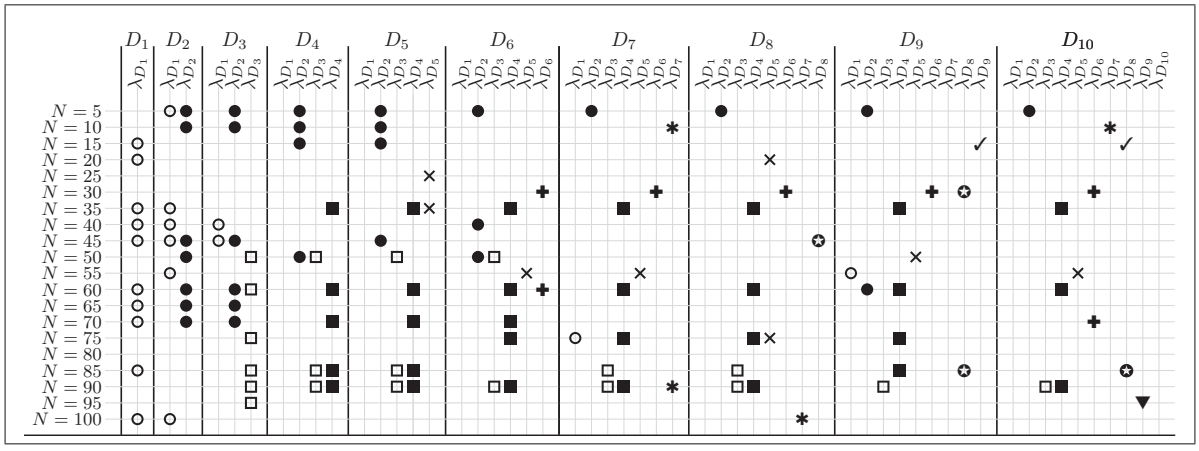


Figure 4.14 Representation of the HMMs selected in Figure 4.13a with the presentation of each new block of data according to the BC_{greedy} algorithm with $tolerance = 0.01$. HMMs trained on different blocks are presented with a different symbol. $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs indicated by the grid on the figure

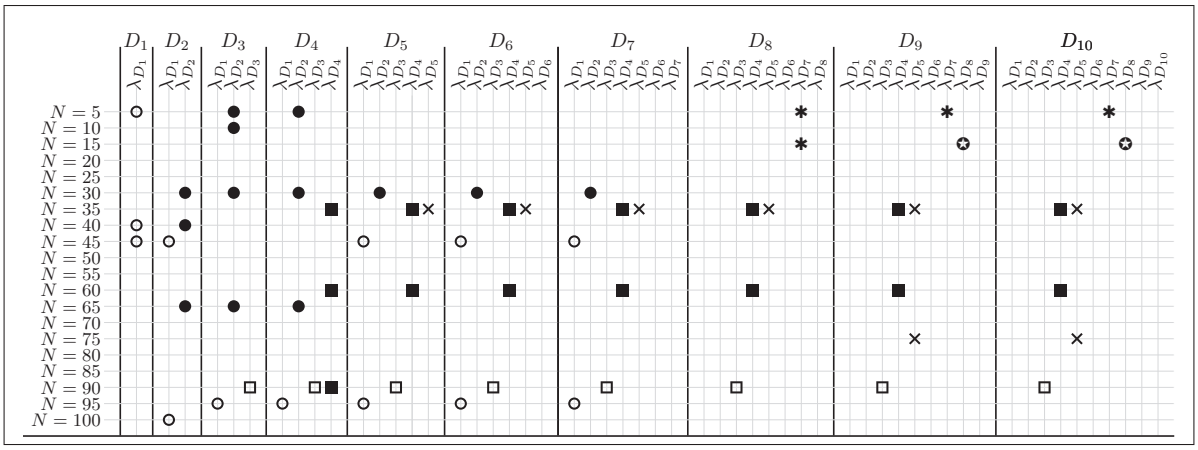


Figure 4.15 Representation of the HMMs selected in Figure 4.13a with the presentation of each new block of data according to the BC_{search} algorithm with $tolerance = 0.01$. HMMs trained on different blocks are presented with a different symbol. $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs indicated by the grid on the figure

Figure 4.13a, presents the results of the BC_{greedy} and BC_{search} algorithms for a tolerated improvement value of 0.01 between the AUCH values. Both BC_{greedy} and BC_{search} are capable of maintaining a slightly lower level of AUCH performance than that of the $incrBC_{all\,N}$ algorithm. However, for each block, the size of the selected EoHMMs ($|E|$) is largely reduced compared with the original pool size ($|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs). The BC_{greedy} algorithm selects ensembles of sizes $|E| = 10$ to 14 HMMs (see Figure 4.14), while BC_{search} selects ensembles of sizes $|E| = 3$ to 8 HMMs (see Figure 4.15). When the number of blocks increases, the number of the selected HMMs according to both algorithms remains almost stable. In particular, at the 10^{th} block of data, BC_{greedy} selects an ensemble of size $|E| = 12$ HMMs, while BC_{search} selects an ensemble of size $|E| = 7$ from the generated pool of size $|\mathcal{P}| = 200$ HMMs, and even provides a slight increase in AUCH performance over that of $incrBC_{all\,N}$. This indicates that both BC_{greedy} and BC_{search} are effective in exploiting the complimentary information provided from the new blocks of data.

As expected, the incremental search strategy employed within the BC_{search} algorithm is most effective for reducing the ensemble sizes while maintaining or improving the overall performance. The selection strategy employed withing the BC_{greedy} algorithm

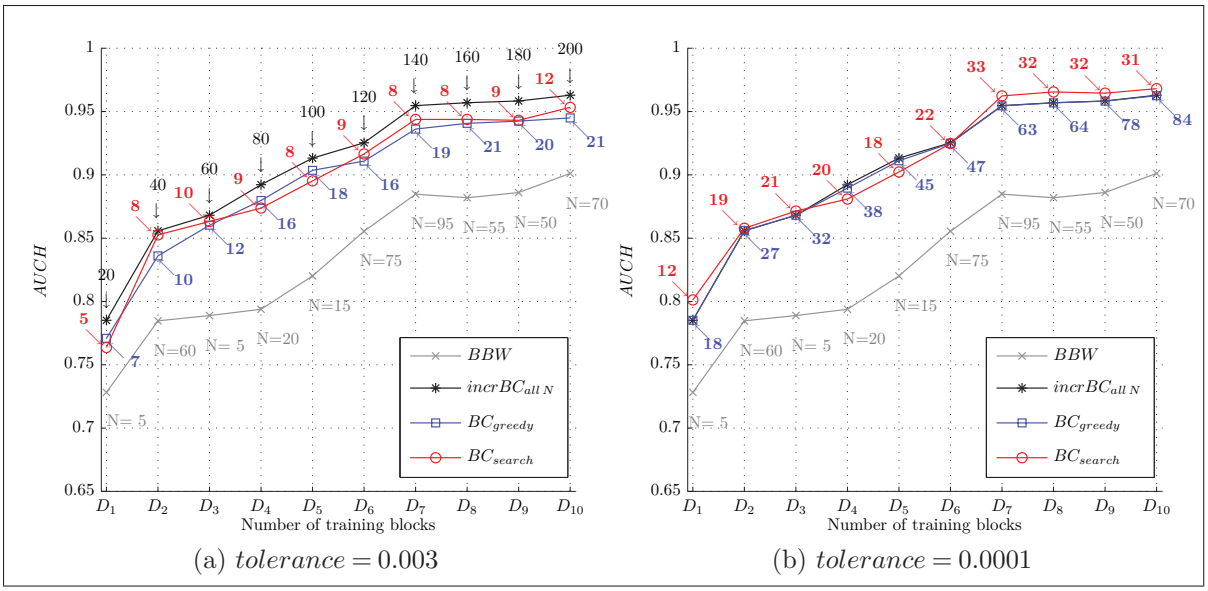


Figure 4.16 Ensemble selection results for sendmail data of Figure 4.12. For each block, the values on the arrows represent the size of the EoHMMs ($|E|$) selected by each technique from $\mathcal{P}$ of size $|\mathcal{P}| = 20, 40, \ldots, 200$ HMMs

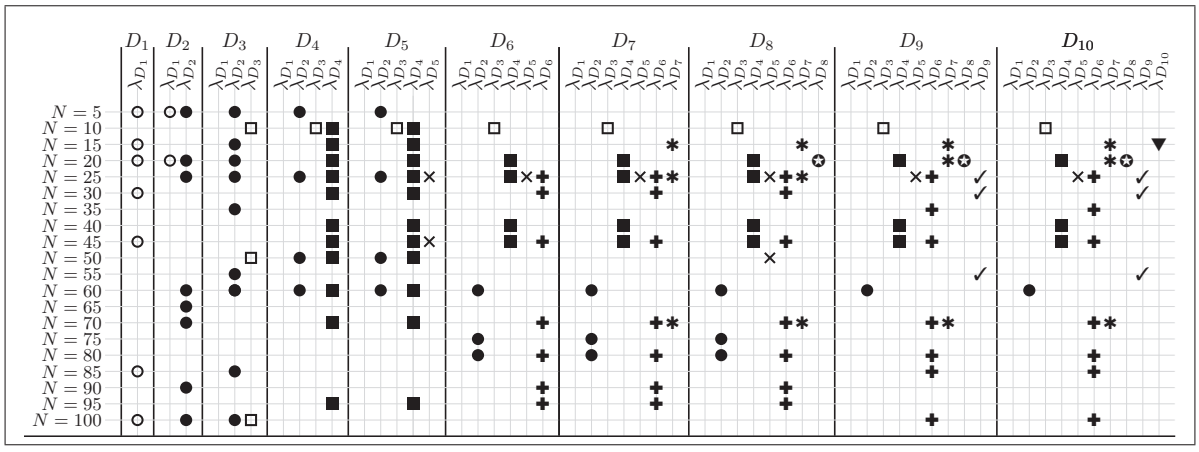


Figure 4.17 Representation of the HMMs selected in Figure 4.16a with the presentation of each new block of data according to the BC_{greedy} algorithm with $tolerance = 0.003$. HMMs trained on different blocks are presented with a different symbol. $|\mathcal{P}| = 20, 40, \ldots 200$ HMMs indicated by the grid on the figure

is more conservative than that of the BC_{search} (see Figure 4.14 and 4.15). BC_{greedy} tends to select and conserve older models due to its linear scanning and selection, while the incremental search strategy of BC_{search} exploits further information by exploring the benefit achieved after combining each new HMM with the cumulative EoHMMs. In

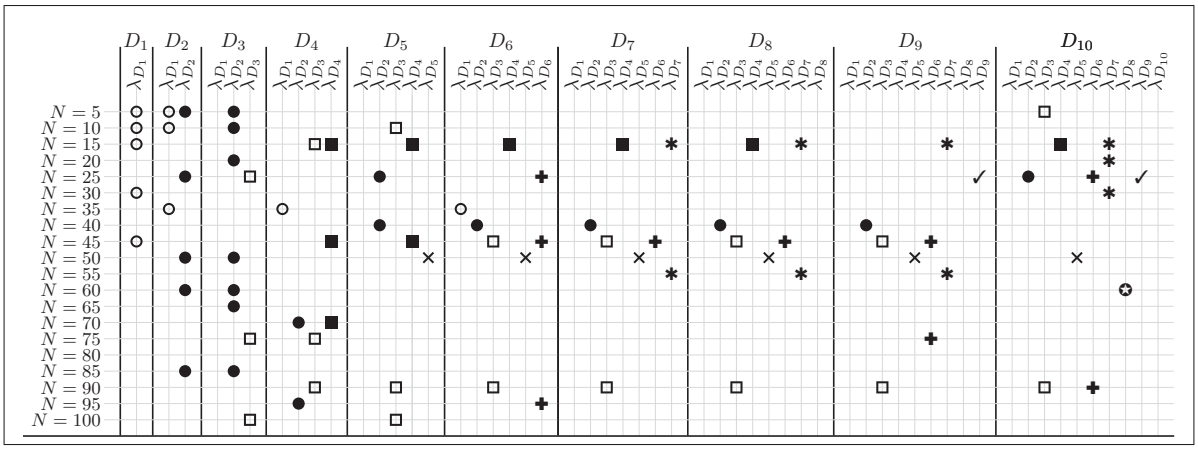


Figure 4.18 Representation of the HMMs selected in Figure 4.16a with the presentation of each new block of data according to the BC_{search} algorithm with $tolerance = 0.003$. HMMs trained on different blocks are presented with a different symbol. $|\mathcal{P}| = 20, 40, \ldots 200$ HMMs indicated by the grid on the figure

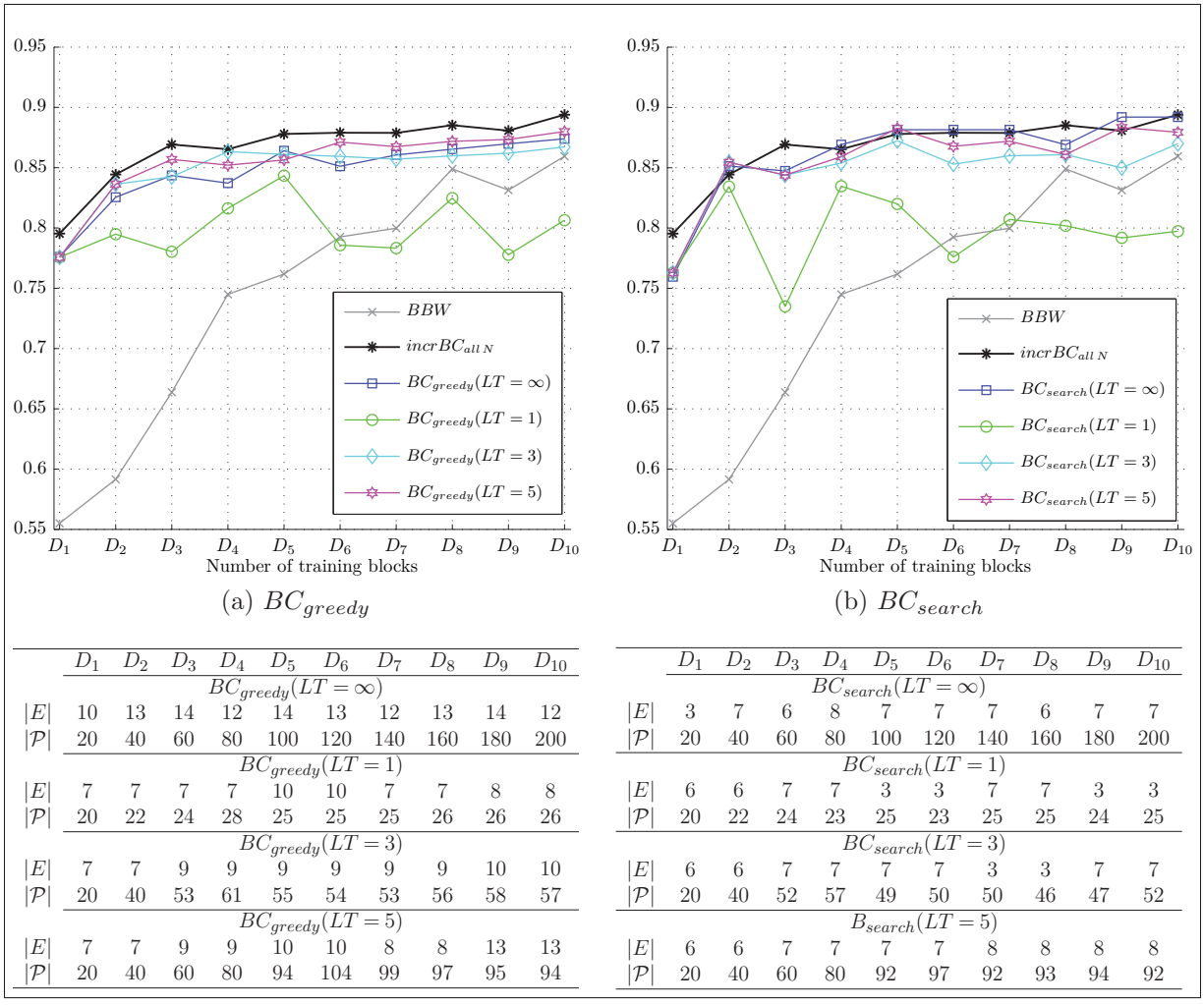
fact, the order in which the models are selected in BC_{search} assures the best ensemble performance up to the tolerance value. Therefore, as illustrated in Figure 4.13b, with lower tolerance values, the level of performance achieved by BC_{search} is higher than that of BC_{greedy} and $incrBC_{all\,N}$ algorithms. In this case, the size of the EoHMMs selected according to BC_{search} is on average about half of that selected by BC_{greedy}, over the ten blocks. However, this comes at the cost of efficiency, where BC_{search} may be an order of magnitude more computationally expensive than BC_{greedy} (see Section 4.3.2). These selection results are also confirmed on sendmail data as shown in Figure 4.16, and detailed in Figure 4.17 and Figure 4.18 for BC_{greedy} and BC_{search} algorithms. In practice, when efficiency is important, BC_{greedy} should be considered as it scans the ordered list of detectors once. Otherwise, BC_{search} has shown to be more effective in selecting detectors that contribute the most to an improved performance of the ensemble.

B. Model Pruning.

Previous experiments have shown that BC_{greedy} and BC_{search} algorithms are effective in maintaining a high level of performance by selecting relatively small sized EoHMMs from the entire pool of previously-generated HMMs. In practice, however, the size of the

pool must be restricted from increasing indefinitely with the number of data blocks. The impact on performance of the pruning strategy proposed in Section 4.3.2 is now evaluated for BC_{greedy} and BC_{search} algorithms according to various life time (LT) expectancy values. In the following results, an HMM is pruned if it is not selected for an LT corresponding to $1, 3$ or 5 data blocks. The reference results of previous experiments, conducted without pruning HMMs from the pool, are shown with an $LT = \infty$.

Figure 4.19 illustrates the impact on accuracy of pruning the pool of HMMs in Figure 4.13a (synthetic data with $\Sigma = 50$ and $CRE = 0.4$), while Figure 4.20 illustrates the impact of pruning the pool of HMMs in Figure 4.16a (sendmail data). The size of the se-



(a) BC_{greedy} (b) BC_{search}

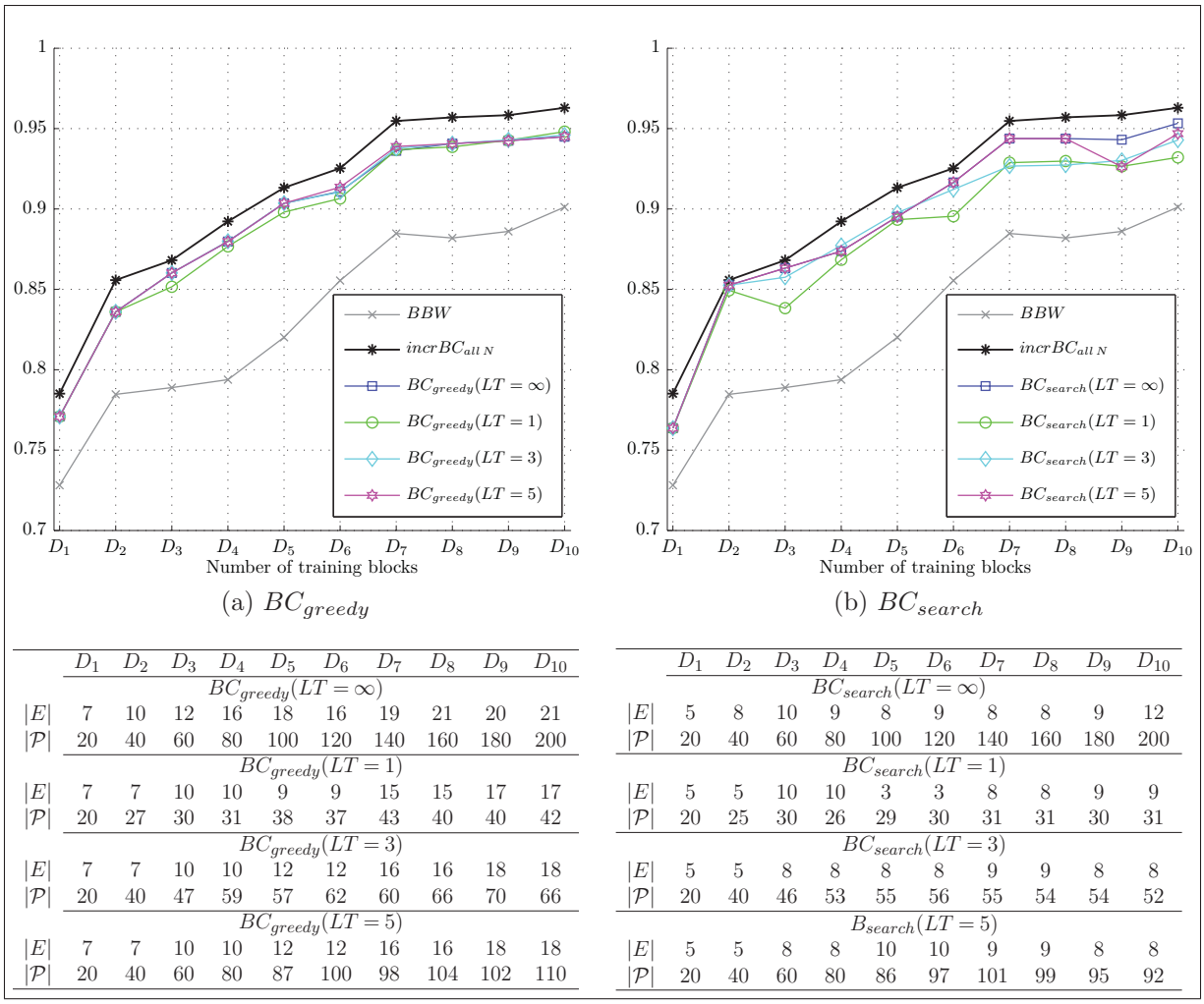
	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{greedy}(LT = \infty)$									
$\|E\|$	10	13	14	12	14	13	12	13	14	12
$\|\mathcal{P}\|$	20	40	60	80	100	120	140	160	180	200
	$BC_{greedy}(LT = 1)$									
$\|E\|$	7	7	7	7	10	10	7	7	8	8
$\|\mathcal{P}\|$	20	22	24	28	25	25	25	26	26	26
	$BC_{greedy}(LT = 3)$									
$\|E\|$	7	7	9	9	9	9	9	9	10	10
$\|\mathcal{P}\|$	20	40	53	61	55	54	53	56	58	57
	$BC_{greedy}(LT = 5)$									
$\|E\|$	7	7	9	9	10	10	8	8	13	13
$\|\mathcal{P}\|$	20	40	60	80	94	104	99	97	95	94

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{search}(LT = \infty)$									
$\|E\|$	3	7	6	8	7	7	7	6	7	7
$\|\mathcal{P}\|$	20	40	60	80	100	120	140	160	180	200
	$BC_{search}(LT = 1)$									
$\|E\|$	6	6	7	7	3	3	7	7	3	3
$\|\mathcal{P}\|$	20	22	24	23	25	23	25	25	24	25
	$BC_{search}(LT = 3)$									
$\|E\|$	6	6	7	7	7	7	3	3	7	7
$\|\mathcal{P}\|$	20	40	52	57	49	50	50	46	47	52
	$B_{search}(LT = 5)$									
$\|E\|$	6	6	7	7	7	7	8	8	8	8
$\|\mathcal{P}\|$	20	40	60	80	92	97	92	93	94	92

Figure 4.19 illustration of the impact on performance of pruning the pool of HMMs in Figure 4.13a

lected EoHMMs ($|E|$) and of the pool ($|\mathcal{P}|$) are presented below the figures for each block of data. The performance of the BBW algorithm and that of $incrBC_{all\,N}$, combining all HMMs without any pruning, are also shown for reference. As shown in Figure 4.19, the level of performance achieved with BC_{greedy} and BC_{search} algorithms is decreased for $LT = 1$ compared to that achieved with $LT = \infty$. This aggressive pruning strategy eliminates all HMMs that have not been selected as ensemble members during the previous incremental learning stage, which may lead to knowledge corruption, and hence degrades the system performance. In fact, the discarded HMMs may have complementary information with respect to the newly-generated HMMs from new block of data. Early elimination of this information limits the search space for future combinations. The impact on performance of pruning depends on the variability of the data and the block size. As shown in Figure 4.20, the decline in performance with $BC_{greedy}(LT = 1)$ and $BC_{search}(LT = 1)$ is relatively small for sendmail data, which incorporates more redundancy than the synthetically-generated data.

The performance achieved with a delayed pruning of HMMs approaches that of retaining all generated HMMs in the pool for larger LT values. As shown in Figures 4.19 and 4.20, the performance achieved by pruning the HMMs that have not been selected for $LT = 3$ and 5 blocks according to BC_{greedy} and BC_{search} algorithms is comparable to that of $BC_{greedy}(LT = \infty)$ and $BC_{search}(LT = \infty)$, respectively. For fixed tolerance and LT values, BC_{search} is capable of selecting smaller EoHMMs and further reducing the size of the pool than BC_{greedy} algorithm. The results show that BC_{search} limits the size of pool to about $LT + 1$ times the averaged number of generated HMMs from new blocks, while BC_{greedy} upper bound on pool size is slightly higher. For instance, with $LT = 5$ we need a storage that accommodates parameters of about 100 HMMs. A fixed-size pool may be obtained by adaptively changing the tolerance and LT values upon receiving a new block of data. In HMM-based ADSs, the system administrator must set these parameters to optimize the system performance, while reducing the size of the selected EoHMMs and of the pool of HMMs for reduced time and memory requirements.



(a) BC_{greedy} (b) BC_{search}

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
$BC_{greedy}(LT=\infty)$										
$\|E\|$	7	10	12	16	18	16	19	21	20	21
$\|\mathcal{P}\|$	20	40	60	80	100	120	140	160	180	200
$BC_{greedy}(LT=1)$										
$\|E\|$	7	7	10	10	9	9	15	15	17	17
$\|\mathcal{P}\|$	20	27	30	31	38	37	43	40	40	42
$BC_{greedy}(LT=3)$										
$\|E\|$	7	7	10	10	12	12	16	16	18	18
$\|\mathcal{P}\|$	20	40	47	59	57	62	60	66	70	66
$BC_{greedy}(LT=5)$										
$\|E\|$	7	7	10	10	12	12	16	16	18	18
$\|\mathcal{P}\|$	20	40	60	80	87	100	98	104	102	110

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
$BC_{search}(LT=\infty)$										
$\|E\|$	5	8	10	9	8	9	8	8	9	12
$\|\mathcal{P}\|$	20	40	60	80	100	120	140	160	180	200
$BC_{search}(LT=1)$										
$\|E\|$	5	5	10	10	3	3	8	8	9	9
$\|\mathcal{P}\|$	20	25	30	26	29	30	31	31	30	31
$BC_{search}(LT=3)$										
$\|E\|$	5	5	8	8	8	8	9	9	8	8
$\|\mathcal{P}\|$	20	40	46	53	55	56	55	54	54	52
$B_{search}(LT=5)$										
$\|E\|$	5	5	8	8	10	10	9	9	8	8
$\|\mathcal{P}\|$	20	40	60	80	86	97	101	99	95	92

Figure 4.20 illustration of the impact on performance of pruning the pool of HMMs in Figure 4.16a

4.6 Conclusion

This chapter presents a ROC-based system to efficiently adapt EoHMMs over time, from new training data, according to a learn-and-combine approach. When a new block of data becomes available, a pool of base HMMs is generated and combined to a global pool comprising previously-generated pools. The base HMMs are trained from each new data block with different number of states and initializations, which allows to capture different underlying structures of the data and hence increases diversity among pool members. The responses from these newly-trained HMMs are then combined with those of the previously-trained HMMs in ROC space using a novel incremental Boolean combination

($incrBC$) technique. On its own, $incrBC$ allows to maintain or improve system accuracy, yet it retains all previously-generated HMMs in the pool.

Specialized model management algorithms are proposed to limit the computational and memory complexity from increasing indefinitely with the $incrBC$ of new HMMs. First, the proposed system selects a diversified EoHMMs from the pool according to one of two ensemble selection algorithms, called BC_{greedy} and BC_{search}, that are adapted to benefit from the monotonicity in $incrBC$ accuracy, for a reduced complexity. Decision thresholds from selected HMMs and Boolean fusion functions are then adapted to improve overall system performance. Finally, redundant and inaccurate base HMMs, which have not been selected for some user-defined time interval, are pruned from the pool. Since the overall composite ROC convex hull is retained, the proposed system is capable of changing its desired operating point during operations, and hence accounts for changes in operating conditions, such as tolerated false alarm rate, prior probabilities, and costs of errors. Most existing techniques for adapting ensembles of classifiers require restarting the training, selection, and combination procedures to account for such a change.

During simulations conducted on both synthetic and real-world HIDS data, the proposed system has been shown to achieve a higher level of accuracy than when parameters of a single best HMM are estimated, at each learning stage, using reference batch and incremental learning techniques. It also outperforms the learn-and-combine approaches using static fusion functions (e.g., majority voting) for combining newly- and previously-generated pools of HMMs. The proposed ensemble selection algorithms have been shown to provide compact and diverse EoHMMs for operations, and hence simplified Boolean fusion rules, while maintaining or improving the overall system accuracy. The employed pruning strategy has been shown to limit the ever-growing pool size, thereby reducing the storage space for accommodating HMMs parameters and the computational costs for the selection algorithms, without negatively affecting the overall system performance.

CONCLUSIONS

Anomaly detection approaches first learn normal system behavior, and then attempt to detect significant deviations from this baseline behavior during operations. Anomaly detection is an active and challenging research area. It has been motivated by numerous real-world applications such as fraud detection in electronic commerce, anomaly detection in control systems and medical image analysis, and target detection in military systems. In computer and network security applications, anomaly detection complements misuse detection techniques for improved intrusion detection systems. Network traffic and protocols, operating systems and host events, as well as user behavior (e.g., keystroke dynamics) can be monitored for anomaly detection. Although capable of detecting novel attacks, ADSs suffer from excessive false alarms and degradation in performance, because they are typically designed using unrepresentative training and validation data with limited prior knowledge about their distributions, and because they face complex environments that change during operations.

Providing representative data for designing ADSs that monitor computer or network events is particularly difficult in today's internetworked environments. An HMM detector employed within an intrusion detection system can not be directly trained from data generated by a process of interest. These data may include patterns of attacks that will be missed by the ADS during operations. As described in Section 1.6.2, unrepresentative normal data is typically provided for training, whether the amount of data is abundant (collected from overly secured environments with restricted functionalities) or limited (collected from unconstrained environments which require significant time and effort to identify existing attacks). The anomaly detector will therefore have an incomplete view of the normal process behavior, and hence rare normal events will be misclassified as anomalous. Therefore, ADSs should be able to efficiently accommodate new data, to account for rare normal events and adapt to changes in normal behaviors that may occur over time.

Contributions and Discussion

This thesis presents novel approaches at the HMM and decision levels for improving the accuracy, efficiency and adaptability of HMM-based ADSs. To sustain a high level of performance, an HMM should be capable to incrementally update its parameters in response to new data that may become available after a it has already been trained on previous data, and deployed for operation. However, the incremental re-estimation of HMM parameters raises several challenges. HMM parameters should be updated from new data without requiring access to the previously-learned training data, and without corrupting previously-acquired knowledge (Grossberg, 1988; Polikar et al., 2001), i.e., previously-learned models of normal behavior. As described in Chapter 2 and Appendix I, standard techniques for estimating HMM parameters involve batch iterative learning, which require a finite amount of training data throughout the training process. The HMM parameters are estimated over several training iterations, where each iteration requires processing the entire training data, until some objective function is maximized. To avoid knowledge corruption, learning new data with these techniques require a cumulative storage of training data over time, and increasingly large time and memory complexity for re-estimation of HMM parameters from the start using all accumulated data. This may represent a very costly solution in practice.

Chapter 2 presents a survey and detailed analysis of on-line learning techniques that have been proposed in literature to estimate HMM parameters from an infinite stream of observation symbols or sub-sequences of observation symbols. They assume being provided with a complete representation of normal system behavior through the stream of generated observations over time, and focus on issues such as efficiency and convergence rate. Therefore, these algorithms update HMM parameters continuously upon observing each new observation symbol or new observation sub-sequence, and typically employ a learning rate to control model stability and accelerate the rate of convergence. When provided with unrepresentative data for training, these on-line learning techniques yield a low level of performance as one pass over the data is not sufficient to capture normal

system behavior. This have been addressed analytically in Chapter 2, and confirmed empirically in Appendix II. When the previously-learned model is considered as a starting point for incremental learning from successive data blocks, both batch and on-line learning algorithms lead to a decline in system performance caused by the plasticity-stability dilemma. In fact these algorithms may remain trapped in local optima of the likelihood function associated with previously-learned data and would not be able to accommodate the newly-acquired information. Otherwise, they will adapt completely to the new data, thereby corrupting the previously-acquired knowledge.

Appendix II extends some on-line learning techniques for HMM parameters to incremental learning, by allowing them to iterate over each block of data and by resetting their internal learning rates when a new block is presented. These learning rates are employed to integrate the information from each new symbol or sub-sequences of observation with the previously-learned model. Evaluation results for detecting system call anomalies, have shown that the extended techniques for incremental learning can provide a higher level of detection accuracy than that of the original on-line learning algorithms, as described in Appendix II. However, optimizing the internal learning rates employed within these algorithms to trade off plasticity against stability remains a challenging task. This is because all previously learned information contained within the current model should be balanced with new information contained in a symbol or sub-sequence of observation.

Appendix III presents an improved incremental learning strategy, which consists of employing an additional (global) learning rate at the iteration level. This aims at delaying the integration of newly-acquired information with the previously-learned model until at least on iteration over the new block of data is performed by the new model. Furthermore, the previously-learned model is kept in memory and weight-balanced with the new model over each iteration. This incremental learning strategy is general in that it can be applied to any batch of on-line learning algorithm. When limited amount of data is provided for training within each new block, batch EM-based algorithms may provide easier solutions, as no internal learning rate is required and monotonic convergence to

a local maximum of the likelihood function is guaranteed. However, when a new block comprising abundant amount of data is provided for incremental learning, on-line algorithms may be employed due to their reduced storage requirements. An optimization of their internal learning rates is yet required to ensure convergence to a local maximum of the likelihood function, although not guaranteed.

The EFFBS algorithm proposed in Appendix I provides an alternative solution, which allow standard batch learning techniques to efficiently estimate HMM parameters from abundant data. In practice, the EFFBS algorithm requires fewer resources than the traditional FB algorithm, yet provides the same results. The memory complexity of FB $\mathcal{O}(NT)$ is reduced to $\mathcal{O}(N)$ with EFFBS, which is independent of the observation sequence length T, but only depends on number of HMM states N. Although EFFBS by itself does not provide an incremental solution for estimation HMM parameters, it can be employed within the BW algorithm according to the incremental learning strategy, when the new block comprises abundant training data. This allows to take advantage of the monotonic convergence over each iteration of the BW algorithm, without any internal learning rate. EFFBS may also be useful in batch learning scenarios, where segmenting a long sequence of observation into smaller training sub-sequences is not desirable. Segmenting long sequences adds another variable to the problem at hand – the best sub-sequence length to be employed, and may raise dependence (or independence) issues between segmented sub-sequences. As discussed in Section 2.2.2.1, training an HMM with multiple sub-sequences assumes their conditional independence (Levinson et al., 1983), or otherwise requires a costly combinatorial techniques Li et al. (2000) to overcome the independence assumption. Training HMM parameters on the entire sequence of system call observations eliminates the issue of sub-sequence length selection during the design phase, and allow to use a desired detector window size for operations (Lane, 2000; Warrender et al., 1999). The issue of learning form long sequences of observations has been also raised in other fields, such as robot navigation systems (Koenig and Simmons, 1996), and bioinformatics (Lam and Meyer, 2010).

Designing an HMM for anomaly detection involves estimating HMM parameters and the number of hidden states, N, from the training data. The value of N and the initial guess of HMM parameters have a considerable impact not only on system accuracy but also on HMM training time and memory requirements, yet overlooked in most related worked as described in Section 1.6. In fact, HMMs trained with different number of states are able to capture different underlying structures of data. Different initializations may lead the algorithm to converge to different solutions in the parameters space due to the many local maxima of the likelihood function. Therefore, a single best HMM will not provide a high level of performance over the entire detection space. Appendix V empirically confirms the impact of N on performance of HMM-based anomaly detectors using different *fixed-size* synthetic HIDS data sets. A multiple-HMMs approach is therefore proposed, where each HMM is trained using a different number of hidden states, and where HMM responses are combined in the ROC space according to the MRROC technique (Scott et al., 1998). Results indicate that this approach provides a higher level of performance than a single best HMM and STIDE matching techniques , over a wide range of training set sizes with various alphabet sizes and irregularity indices, and different anomaly sizes.

Chapter 3 presents a novel iterative Boolean combination (*IBC*) technique for efficient fusion of the responses from multiple classifiers in the ROC space. The *IBC* efficiently exploits *all* Boolean functions applied to the ROC curves and therefore requires no prior assumptions about conditional independence of detectors or convexity of ROC curves. During simulations conducted on *fixed-size* synthetic and real HIDS data sets, the *IBC* has been also applied to combine the responses a of multiple-HMM system, where each HMM is trained using a different number of states and initializations. Results have shown that, even with one iteration, the *IBC* technique always increases system performance over related ROC-based combination techniques, such as the MRROC fusion, and the Boolean conjunction or disjunction fusion functions, without a significant computational and storage overhead. In fact, the time complexity of *IBC* is linear with the number of classifiers while its memory requirement is independent of the number of classifiers,

which allows for a large number of combinations. When the *IBC* is allowed to iterate until convergence, the performance improves significantly while the time and memory complexity required for each iteration are reduced by an order of magnitude with reference to the first iteration. The performance gain, especially when provided with limited training data, is due to the ability of the *IBC* technique to exploit diverse information residing in inferior points on the ROC curves, which are disregarded by the other techniques. *IBC* has been also shown useful for repairing the concavities in a ROC curve. The proposed *IBC* is general in that it can be employed to combine diverse responses of any crisp or soft detectors, within a wide range of application domains. For instance, it has been successfully applied to combine the responses from different biometric systems (trained on the same *fixed-size* data) for iris recognition (Gorodnichy et al., 2011).

Similar to batch learning form fixed-size data sets, a single HMM system for incremental learning from successive blocks of data may not approximate the underlying data distribution adequately. Chapter 4 presents an adaptive system for incremental learning of new data over time using a learn-and-combine approach based on the proposed Boolean combination techniques. When a new block of training data becomes available, a new pool of HMMs is generated using different number of HMM states and random initializations. The responses from the newly-trained HMMs are then incrementally combined to those of the previously-trained HMMs in ROC space using the proposed Boolean combination techniques. Since the pool size grows indefinitely as new blocks of data become available over time, employing specialized model management strategies is therefore a crucial aspect for maintaining the efficiency of an adaptive ADS. The proposed ensemble selection techniques have been shown to form compact EoHMMs by selecting diverse and accurate base HMMs from the pool, while maintaining or improving the overall system accuracy. The proposed pruning strategy has been shown to limit the pool size from increasing indefinitely with the number of data blocks, reducing thereby the storage space for accommodating HMMs parameters and the computational costs of the selection algorithms, without negatively affecting the overall system performance. The proposed

system is capable of changing its desired operating point during operations, and this point can be adjusted to changes in prior probabilities and costs of errors.

During simulations conducted for incremental learning from on successive blocks of synthetic and real-world HIDS data, the proposed learn-and-combine approach has been shown to achieve the highest level of accuracy than all related techniques presented in Chapter 4. In particular, it has been shown to achieve higher level of accuracy than when parameters of a single best HMM are estimated, at each learning stage, using reference batch (BBW) and the proposed incremental learning techniques (IBW).

The results of this thesis provide an answer to the main research question: *Given newly-acquired training data, is it best to adapt parameters of HMM detectors trained on previously-acquired data, or to train new HMMs on newly-acquired data and combine them with those trained on previously-acquired data?* Adapting parameters of a previously-trained (single) HMM to newly-acquired training data according to IBW provide an efficient solution, which can limit existing knowledge corruption, and maintain or improve the detection accuracy over time. However, the corruption of existing knowledge caused by the large number of local maxima in the solution space remains an issue. This is illustrated in the large variances of the results achieved with the IBW algorithm, and in the lower level of detection accuracy compared to that of BBW (see Section 4.5).

In this thesis, a learn-and-combine has been proposed whereby a new HMM is trained on newly-acquired data and its responses are combined with those trained on previously-acquired data using incremental Boolean combination in ROC space. This approach has been shown to provide the overall highest level of accuracy over time, especially when training data is limited (see Section 4.5). In contrast, existing techniques for combining HMMs have been shown shown unable to maintain system accuracy over time. For instance, the combination of these (previously- and newly-trained) HMMs at the detector level by using model averaging or weighted averaging (Hoang and Hu, 2004), have been shown inadequate for ergodic HMMs and lead to knowledge corruption, as presented in

Section II.5. Similarly, combining the outputs of these HMMs using traditional fusion techniques at the score-level (such as median) or decision-level (e.g., majority vote), have been shown to provide poor detection accuracy over time (see Section 4.5). The learn-and-combine approach may require more resources than IBW since several HMMs are employed during operations. Depending on the operation requirements, higher tolerance values can be imposed on the model selection algorithms to reduce the number of selected HMMs for operations, and hence trade-off accuracy with efficiency. Finally, the learn-and-combine approach provide a general and versatile solution for adapting and selecting HMMs over time. In particular, HMMs that are incrementally updated according to IBW can be also combined according to the learn-and-combine approach.

Several techniques have been applied to learn the normal process behavior using system call sequences (Forrest et al., 2008). It is outside the scope of this thesis to compare with every possible approach. In particular, different experimental protocols (data pre-processing, detector window size, detector training, etc.) and different evaluation techniques (anomaly definition and labeling) have been employed during the conducted experiments. Therefore, comparison, in this thesis have been limited to classical sequence matching techniques, HMM batch and incremental techniques, and all related fusion techniques.

Future work

Several works have discussed the possibility of evading system call based IDSs by disguising anomalous sub-sequences within the normal process behavior. Mimicry attacks (Krügel et al., 2005; Parampalli et al., 2008; Tandon and Chan, 2006; Wagner and Soto, 2002) rely on crafting code that can generate malicious system call sequences that are considered legitimate by anomaly detection model. For instance, changing foreign system calls required for the attack to equivalent system calls used by the original process (Tan et al., 2002). In essence, mimicry attacks illustrate possible disadvantages of using the system call temporal order only as features. Therefore, some authors proposed including

system call arguments additional features (Cho and Han, 2003; Tandon and Chan, 2006). The applicability of mimicry attacks and their variants in practice, and the techniques to detect these attacks are addressed by Forrest et al. (2008). The proof-of-concept simulations conducted in this thesis considered the order of system calls only, for simplicity and availability of UNM datasets. However, including feature vectors to train HMM is a direct extension and does not affect the proposed incremental learning strategy nor the EFFBS algorithm. In reality there are many different types of intrusions, and different features and detectors are therefore needed.

A hybrid intrusion detection system combines two or more types of IDSs for improved accuracy. A hybrid IDS attempts to provide a more comprehensive system that may address the limitations of each IDS type. For instance, anomaly detection techniques may provide useful information to define signatures for misuse detectors, yielding an improved classification of attacks. On the other hand, misuse detection systems provide precise and descriptive alarms; while anomaly detection can only detect symptoms of attacks without specific knowledge of details. A hybrid IDS can have both host and network sensors and thus detects both host-based and network-based attacks. Furthermore, a HIDS may complement a NIDS as a verification system, to confirm for instance whether the suspicious network traffic has resulted in a successful attack or not.

In this research, the proposed Boolean combination techniques are applied to combine responses from the same one-class classifiers (HMMs) trained on the same data and using the same features, however according to different hyperparameters and initializations. Future work involves applying these techniques to combine various detection techniques employed within a hybrid IDS. For instance, to combine the response from the same detectors trained using different feature sets and also to combine the responses from different detectors (e.g., using supervised, unsupervised, generative, and discriminative approaches) trained on the same data. Investigation of diversity measures and impact of correlations within the ROC space is an important future direction. This may yield

further insight into the design of classifier ensembles that contribute toward efficient and accurate combinations.

The robustness of the learn-and-combine approach depends on maintaining a representative validation set over time, for selection of HMMs, decision thresholds and Boolean functions. The proposed ADS adopts a human-centric approach, which relies on the system administrator to update the validation set with recent and most informative anomalous sub-sequences. Future work involves investigating techniques such as active learning to reduce labeling and selection costs. The presented results are for validation and test sets that comprised moderate skew (up to 75% of normal and 25% of anomalous data). Another future investigation would be to assess the impact on system performance of heavily imbalanced data.

Another important issue is the ability to operate in dynamically-changing environments. In such cases, some novelty criteria on the new data should be employed to detect changes and trigger adaptation by resting a monotonically decreasing learning rate or fine-tuning an auto-adaptive learning rate. It may be useful to investigate whether the on-line algorithms for estimation of HMM parameters, which are surveyed in Chapter 2, can be employed to detect changes in normal behaviors over time.

Although not explored in this thesis, the proposed learn-and-combine approach can handle concept drift in changing environments, which is typically attributed to changes in prior, class-conditional, or posterior probability distribution of classes (Kuncheva, 2004b). As designed, the learn-and-combine approach allows to account for changes in class prior probabilities. Recent techniques for on-line estimation of evolving class priors (Zhang and Zhou, 2010) can, thereby, be directly employed in the proposed system, for a dynamic selection of the best ensemble of classifiers and for adaptation of decision thresholds and Boolean fusion functions. Other types of drift can be accounted for by updating training data and ensemble members, replacing inaccurate classifiers, and adding new features (Kuncheva, 2004b). The learn-and-combine approach is designed to select and combine

output decisions from different homogeneous or heterogeneous, crisp or soft detectors, which are trained on different data or features using on-line or batch learning techniques. Another interesting future work is to evaluate the ability of the learn-and-combine approach to adapt to concept drift and evolving priors in various detection applications with dynamically changing environments.

APPENDIX I

ON THE MEMORY COMPLEXITY OF THE FORWARD BACKWARD ALGORITHM*

Abstract

The Forward-Backward (FB) algorithm forms the basis for estimation of Hidden Markov Model (HMM) parameters using the Baum-Welch technique. It is however known to be prohibitively costly when estimation is performed from long observation sequences. Several alternatives have been proposed in literature to reduce the memory complexity of FB at the expense of increased time complexity. In this paper, a novel variation of the FB algorithm – called the Efficient Forward Filtering Backward Smoothing (EFFBS) – is proposed to reduce the memory complexity without the computational overhead. Given an HMM with N states and an observation sequence of length T, both FB and EFFBS algorithms have the same time complexity, $\mathcal{O}(N^2T)$. Nevertheless, FB has a memory complexity of $\mathcal{O}(NT)$, while EFFBS has a memory complexity that is independent of T, $\mathcal{O}(N)$. EFFBS requires fewer resources than FB, yet provides the same results.

I.1 Introduction

Hidden Markov Model (HMM) is a stochastic model for sequential data. Provided with an adequate number of states and sufficient set of data, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in term of simple and compact models. The forward-backward (FB) algorithm is a dynamic programming technique that forms the basis for estimation of HMM parameters. Given a finite sequence of training data, it efficiently evaluates the likelihood of this data given an HMM, and computes the smoothed conditional state probability densities – the sufficient statistics – for updating HMM parameters according to the Baum-Welch (BW) algorithm

*This chapter is published as an article in Pattern Recognition Letters journal, DOI:10.1016/j.patrec.2009.09.023

(Baum, 1972; Baum et al., 1970). BW is an iterative expectation maximization (EM) technique specialized for batch learning of HMM parameters via maximum likelihood estimation.

The FB algorithm is usually attributed to Baum (1972); Baum et al. (1970), although it was discovered earlier by Chang and Hancock (1966) and then re-discovered in different areas in the literature (Ephraim and Merhav, 2002). Despite suffering from numerical instability, the FB remains more famous than its numerically stable counterpart; the Forward Filtering Backward Smoothing (FFBS)[1] algorithm (Ott, 1967; Raviv, 1967). As a part of the BW algorithm, the FB algorithm has been employed in various applications such as signal processing, bioinformatics and computer security, (Cappe, 2001).

When learning from long sequences of observations the FB algorithm may be prohibitively costly in terms of time and memory complexity. This is the case, for example, of anomaly detection in computer security (Lane, 2000; Warrender et al., 1999), protein sequence alignment (Krogh et al., 1994) and gene-finding (Meyer and Durbin, 2004) in bioinformatics, and robot navigation systems (Koenig and Simmons, 1996). For a sequence of length T and an ergodic HMM with N states, the memory complexity of the FB algorithm grows linearly with T and N, $\mathcal{O}(NT)$, and its time complexity grows quadratically with N and linearly with T, $\mathcal{O}(N^2T)$ [2]. For a detailed analysis of complexity the reader is referred to Table AI.1 (Section I.3). When T is very large, the memory complexity may exceed the resources available for the training process, causing overflow from internal system memory to disk storage.

Several alternatives have been proposed in literature trying to reduce the memory complexity of the FB algorithm or, ideally, to eliminate its dependence on the sequence length T. Although there is an overlap between these approaches and some have been re-discovered, due to the wide range of HMMs applications, they can be divided into

[1] FFBS is also referred to in literature as $\alpha-\gamma$ and disturbance smoothing algorithm.

[2] If the HMM is not fully connected, the time complexity reduces to $\mathcal{O}(NQ_{max}T)$, where Q_{max} is the maximum number of states that any state is connected to.

two main types. The first type performs exact computations of the state densities using fixed-interval smoothing (similar to FB and FFBS), such as the checkpointing and forward-only algorithms. However, as detailed in section I.3, the memory complexity of the former algorithm still depends on T, while the latter eliminates this dependency at the expenses of a significant increase in time complexity, $\mathcal{O}(N^4T)$.

Although outside the scope of this paper, there are approximations to the FB algorithm. Such techniques include approximating the backward variables of the FB algorithm using a sliding time-window on the training sequence (Koenig and Simmons, 1996), or slicing the sequence into several shorter ones that are assumed independent, and then applying the FB on each sub-sequence (Wang et al., 2004; Yeung and Ding, 2003). Other solutions involve performing on-line learning techniques on a finite data however (Florez-Larrahondo et al., 2005; LeGland and Mevel, 1997). These techniques are not explored because they provide approximations to the smoothed state densities, and hence lead to different HMMs. In addition, their theoretical convergence properties are based on infinite data sequence ($T \rightarrow \infty$).

A novel modification to the FFBS called Efficient Forward Filtering Backward Smoothing (EFFBS) is proposed, such that its memory complexity is independent of T, without incurring a considerable computational overhead. In contrast with the FB, it employs the inverse of HMM transition matrix to compute the smoothed state densities without any approximations. A detailed complexity analysis indicates that the EFFBS is more efficient than existing approaches when learning of HMM parameters form long sequences of training data.

This paper is organized as follows. In Section I.2, the estimation of HMM parameters using the FB and the Baum-Welch algorithms is presented. Section I.3 reviews techniques in literature that addressed the problem of reducing the memory complexity of the FB algorithm, and accesses their impact on the time complexity. The new EFFBS algorithm is presented in Section I.4, along with a complexity analysis and case study.

I.2 Estimation of HMM Parameters

The Hidden Markov Model (HMM) is a stochastic model for sequential data. A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, and a set of observation probability distributions, each one associated with a state. At each discrete time instant, the process is assumed to be in a state, and an observation is generated by the probability distribution corresponding to the current state. The model is termed *discrete* (or finite-alphabet) if the output alphabet is finite. It is termed *continuous* (or general) if the output alphabet is not necessarily finite e.g., the state is governed by a parametric density function, such as Gaussian, Poisson, etc. For further details regarding HMM the reader is referred to the extensive literature (Ephraim and Merhav, 2002; Rabiner, 1989).

Formally, a discrete-time finite-state HMM consists of N hidden states in the finite-state space $S = \{S_1, S_2, ..., S_N\}$ of the Markov process. Starting from an initial state S_i, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij} $(1 \leq i, j \leq N)$. As illustrated in Figure AI.1, the process then emits a symbol v according to the output probability distribution $b_j(v)$, which may be discrete or continuous, of the current state S_j. The model is therefore parametrized by the set $\lambda = (\pi, A, B)$, where vector $\pi = \{\pi_i\}$ is initial state probability distribution, matrix $A = \{a_{ij}\}$ denotes the state transition probability distribution, and matrix $B = \{b_j(k)\}$ is the state output probability distribution.

I.2.1 Baum-Welch Algorithm:

The Baum-Welch (BW) algorithm (Baum and Petrie, 1966; Baum et al., 1970) is an Expectation-Maximization (EM) algorithm (Dempster et al., 1977) specialized for estimating HMM parameters. It is an iterative algorithm that adjusts HMM parameters to best fit the observed data, $o_{1:T} = \{o_1, o_2, \ldots, o_T\}$. This is typically achieved by maximiz-

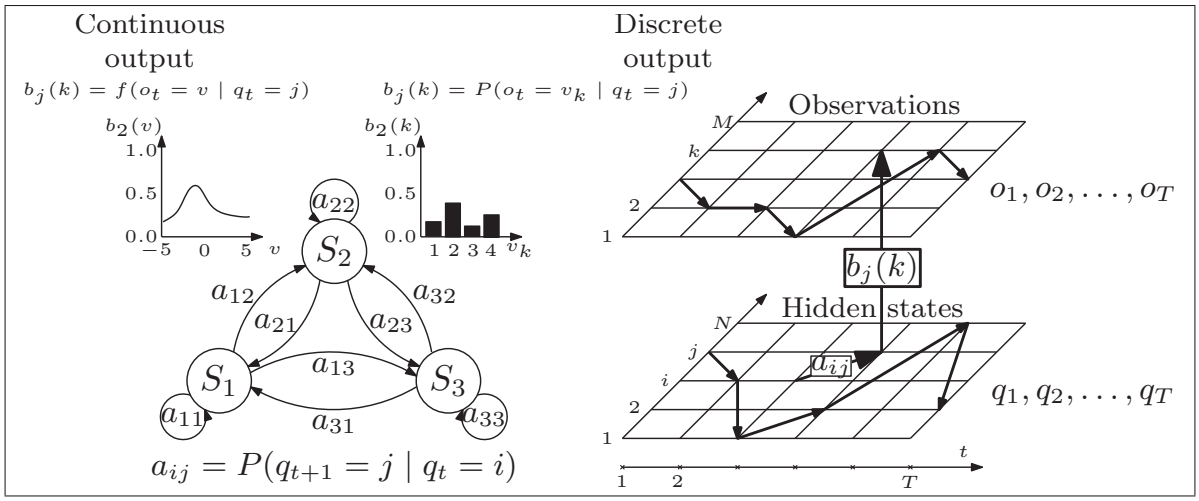


Figure AI.1 Illustration of a fully connected (ergodic) three states HMM with either a continuous or discrete output observations (left). Illustration of a discrete HMM with N states and M symbols switching between the hidden states q_t and generating the observations o_t (right). $q_t \in S$ denotes the state of the process at time t with $q_t = i$ means that the state at time t is S_i

ing the log-likelihood, $\ell_T(\lambda) \triangleq \log P(o_{1:T} \mid \lambda)$ of the training data over HMM parameters space (Λ):

$$\lambda^* = \text{argmax}_{\lambda \in \Lambda} \ell_T(\lambda) \tag{I.1}$$

The subscripts $(.)_{t|T}$ in formulas (II.5) to (II.8) are used to stress the fact that these are *smoothed* probability estimates. That is, computed from the whole sequence of observations during batch learning.

During each iteration, the E-step computes the sufficient statistics, i.e., the *smoothed a posteriori* conditional state density:

$$\gamma_{t|T}(i) \triangleq P(q_t = i \mid o_{1:T}, \lambda) \tag{I.2}$$

and the *smoothed a posteriori* conditional joint state density [3]:

$$\xi_{t|T}(i,j) \triangleq P(q_t = i, q_{t+1} = j \mid o_{1:T}, \lambda) \tag{I.3}$$

[3]To facilitate reading, the terms a posteriori and conditional will be omitted whenever it is clear from the context

which are then used, in the M-step, to re-estimate the model parameters:

$$\begin{aligned} \pi_i^{(k+1)} &= \gamma_{1|T}^{(k)}(i) \\ a_{ij}^{(k+1)} &= \frac{\sum_{t=1}^{T-1} \xi_{t|T}^{(k)}(i,j)}{\sum_{t=1}^{T-1} \gamma_{t|T}^{(k)}(i)} \\ b_j^{(k+1)}(m) &= \frac{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j) \delta_{o_t v_m}}{\sum_{t=1}^{T} \gamma_{t|T}^{(k)}(j)} \end{aligned} \tag{I.4}$$

The Kronecker delta δ_{ij} is equal to one if $i = j$ and zero otherwise.

Starting with an initial guess of the HMM parameters, λ^0, each iteration k, of the E- and M-step is guaranteed to increase the likelihood of the observations giving the new model until a convergence to a stationary point of the likelihood is reached (Baum, 1972; Baum et al., 1970).

The objective of the E-step is therefore to compute an estimate $\hat{q}_{t|\tau}$ of an HMM's hidden state at any time t given the observation history $o_{1:\tau}$. The optimal estimate $\hat{q}_{t|\tau}$ in the minimum mean square error (MMSE) sense of the state q_t of the HMM at any time t, given the observation is the conditional expectation of the state q_t given $o_{1:\tau}$ (Li and Evans, 1992; Shue et al., 1998):

$$\hat{q}_{t|\tau} = E(q_t \mid o_{1:\tau}) = \sum_{i=1}^{N} \gamma_{t|\tau}(i) \tag{I.5}$$

In estimation theory, this conditional estimation problem is called *filtering* if $t = \tau$; *prediction* if $t > \tau$, and *smoothing* if $t < \tau$. Figure AI.2 provides an illustration of these estimation problems. The smoothing problem is termed *fixed-lag* smoothing when computing the $E(q_t \mid o_{1:t+\Delta})$ for a fixed-lag $\Delta > 0$, and *fixed-interval* smoothing when computing the $E(q_t \mid o_{1:T})$ for all $t = 1, 2, \ldots, T$.

For batch learning, the state estimate is typically performed using fixed-interval smoothing algorithms (as described in Subsections I.2.2 and I.2.3). Since it incorporates more

evidence, fixed-interval smoothing provides better estimate of the states than filtering. This is because the latter relies only on the information that is available at the time.

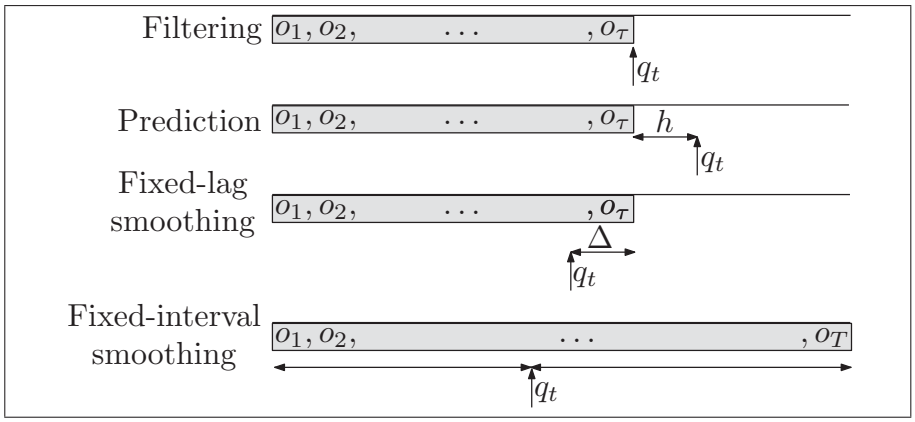


Figure AI.2 Illustration of the filtering, prediction and smoothing estimation problems. Shaded boxes represent the observation history (already observed), while the arrows represent the time at which we would like to compute the state estimates

I.2.2 Forward-Backward (FB)

The Forward-Backward (Baum, 1972; Baum et al., 1970; Chang and Hancock, 1966) is the most commonly used algorithm for computing the smoothed state densities of Eqs. (II.5) and (II.6). Its forward pass computes the *joint* probability of the state at time t *and* the observations up to time t:

$$\alpha_t(i) \triangleq P(q_t = i, o_{1:t} \mid \lambda), \tag{I.6}$$

using the following recursion, for $t = 1, 2, \ldots, T$ and $j = 1, \ldots, N$:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1}) \tag{I.7}$$

Similarly, the backward path computes the probability of the observations from time $t+1$ up to T *given* the state at time t,

$$\beta_t(i) \triangleq P(o_{t+1:T} \mid q_t = i, \lambda), \tag{I.8}$$

according to the reversed recursion, for $t = T-1, \ldots, 1$ and $i = 1, \ldots, N$:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{I.9}$$

Then, both smoothed state and joint state densities of Eqs. (II.5) and (II.6) – the sufficient statistics for HMM parameters update – are directly obtained:

$$\gamma_{t|T}(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^{N} \alpha_t(k)\beta_t(k)} \tag{I.10}$$

$$\xi_{t|T}(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_t(k) a_{kl} b_l(o_{t+1}) \beta_{t+1}(l)} \tag{I.11}$$

By definition, the elements of α are not probability measures unless normalized, that is why they are usually called the α-variables. Due to this issue, the FB algorithm is not stable and susceptible to underflow when applied to a long sequence of observation. In Levinson et al. (1983) and Rabiner (1989) the authors suggest scaling the α-variables by their summation at each time:

$$\bar{\alpha}_t(i) \quad = \quad \frac{\alpha_t(i)}{\sum_i \alpha_t(i)} = \frac{P(q_t = i, o_{1:t}\lambda)}{P(o_{1:t}\lambda)} = P(q_t = i \mid o_{1:t}, \lambda) \tag{I.12}$$

making for a *filtered* state estimate. This of course requires scaling the beta β values, which have no an intuitive probabilistic meaning, with the same quantities (refer to Algorithm AI.1 and AI.2). Other solutions involves working with extended exponential and extended logarithm functions (Mann, 2006).

I.2.3 Forward Filtering Backward Smoothing (FFBS)

Algorithm AI.1: Forward-Scaled($o_{1:T}, \lambda$)

Output: $\alpha_t(i) \triangleq P(q_t = i, o_{1:t} \mid \lambda)$, $\ell_T(\lambda)$ and $scale()$

for $i = 1, \ldots, N$ **do** // Initialization
  $\alpha_1(i) \leftarrow \pi_i b_i(o_1)$
$scale(1) \leftarrow \sum_{i=1}^{N} \alpha_1(i)$
for $i = 1, \ldots, N$ **do**
  $\alpha_1(i) \leftarrow \alpha_1(i)/scale(1)$
for $t = 1, \ldots T-1$ **do** // Induction
  for $j = 1, \ldots, N$ **do**
    $\alpha_{t+1}(j) \leftarrow \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1})$
  $scale(t+1) \leftarrow \sum_{i=1}^{N} \alpha_{t+1}(i)$
  for $i = 1, \ldots, N$ **do**
    $\alpha_{t+1}(i) \leftarrow \alpha_{t+1}(i)/scale(t+1)$
for $t = 1, \ldots T$ **do** // Evaluation
  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + log(scale(t))$

Algorithm AI.2: Backward-Scaled ($o_{1:T}, \lambda, scale()$)

Output: $\beta_t(i) \triangleq P(o_{t+1:T} \mid q_t = i, \lambda)$
for $i = 1, \ldots, N$ **do** // Initialization
  $\beta_T(i) \leftarrow 1$
for $t = T-1, \ldots, 1$ **do** // Induction
  for $i = 1, \ldots, N$ **do**
    $\beta_t(i) \leftarrow \frac{\sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{scale(t+1)}$

As highlighted in literature (Devijver, 1985; Ephraim and Merhav, 2002; Cappe and Moulines 2005), the FFBS is a lesser known but numerically *stable* alternative algorithm to FB, which has been proposed in Askar and Derin (1981); Lindgren (1978); Ott (1967); Raviv (1967). In addition, the FFBS is probabilistically more meaningful since it propagates probability densities in its forward and backward passes.

During the forward pass, the algorithm splits the state density estimates into *predictive* (I.13) and *filtered* (I.14) state densities:

$$\gamma_{t|t-1}(i) \triangleq P(q_t = i \mid o_{1:t-1}, \lambda) \tag{I.13}$$

$$\gamma_{t|t}(i) \triangleq P(q_t = i \mid o_{1:t}, \lambda) \tag{I.14}$$

The initial predictive state density is the the initial probability distribution, $\gamma_{1|0}(i) = \pi_i$. For $t = 1, \ldots, T$, the filtered state density at time t is computed from the predictive state density at time $t-1$,

$$\gamma_{t|t}(i) = \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{j=1}^{N} \gamma_{t|t-1}(j) b_j(o_t)} \tag{I.15}$$

and then, the predictive state density at time $t+1$ is computed from the filtered state density at time t:

$$\gamma_{t+1|t}(j) = \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij} \tag{I.16}$$

Both the filtered and one-step prediction density, follow from a combination of the Bayes' rule of conditional probability and the Markov property.

Algorithm AI.3: Forward-Filtering($o_{1:T}, \lambda$)

Output: $\gamma_{t|t}(i) = P(q_t = i \mid o_{1:t}, \lambda)$ and $\ell_T(\lambda)$

for $i = 1, \ldots, N$ **do** // Initialization
  $\gamma_{1|0}(i) = \pi_i$
for $t = 1, \ldots T-1$ **do** // Induction
  for $i = 1, \ldots, N$ **do**
    $\gamma_{t|t}(i) \leftarrow \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{j=1}^{N} \gamma_{t|t-1}(j) b_j(o_t)}$
  for $j = 1, \ldots, N$ **do**
    $\gamma_{t+1|t}(j) \leftarrow \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}$
for $t = 1, \ldots T$ **do** // Evaluation
  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + \log \sum_{i=1}^{N} b_i(o_t) \gamma_{t|t-1}(i)$

Algorithm AI.4: Backward-Smoothing ($\gamma_{T|T}, \lambda$)

Output: $\gamma_{t|T}(i) = P(q_t = i \mid o_{1:T})$ and $\xi_{t|T}(i,j) = P(q_t = i, q_{t+1} = j \mid o_{1:T})$

$\gamma_{T|T} \leftarrow input$ // Initialization
for $t = T-1, \ldots, 1$ **do** // Induction
  for $i = 1, \ldots, N$ **do**
    for $j = 1, \ldots, N$ **do**
      $\xi_{t|T}(i,j) \leftarrow \frac{\gamma_{t|t}(i) a_{ij}}{\sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}} \gamma_{t+1|T}(j)$
    $\gamma_{t|T}(i) \leftarrow \sum_{j=1}^{N} \xi_{t|T}(i,j)$

Backward recursion, leads directly to the required *smoothed* densities by solely using the filtered and predictive state densities (Askar and Derin, 1981; Lindgren, 1978) for $t = T-1, \ldots, 1$:

$$\xi_{t|T}(i,j) = \frac{\gamma_{t|t}(i)a_{ij}}{\sum_{i=1}^{N}\gamma_{t|t}(i)a_{ij}}\gamma_{t+1|T}(j) \tag{I.17}$$

$$\gamma_{t|T}(i) = \sum_{j=1}^{N}\xi_{t|T}(i,j) \tag{I.18}$$

In contrast with the FB algorithm, the backward pass of the FFBS does *not* require access to the observations. In addition, since it is propagating probabilities densities, in both forward and backward passes, the algorithm is numerically stable. In the forward pass, the *filtered* state density can be also computed in the same way as the normalized $\bar{\alpha}$-variables (I.12), since after the normalization $\bar{\alpha}$ becomes the state filter. When the forward pass is completed at time T, $\gamma_{T|T}$ is the only *smoothed* state density, while all previous ones are filtered and predictive state estimates. The reader is referred to Algorithm AI.3 and AI.4 for further details on the FFBS.

I.3 Complexity of Fixed-Interval Smoothing Algorithms

This section presents a detailed analysis of the time and memory complexity for the FB and FFBS algorithms. It also presents the complexity of the checkpointing and forward only algorithms. These algorithms have been proposed to reduce the memory complexity of FB at the expense of time complexity.

The time complexity is defined as the sum of the worst-case running time for each operation (e.g., multiplication, division and addition) required to process an input. The growth rate is then obtained by making the parameters of the worst-case complexity tend to ∞. Memory complexity is estimated as the number of 32 bit registers needed during learning process to store variables. Only the worst-case memory space required during processing phase is considered.

I.3.1 FB and FFBS Algorithms

The main bottleneck for both FB and FFBS algorithms occurs during the induction phase. In the ergodic case, the time complexity per iteration is $\mathcal{O}(N^2T)$ as shown in (I.7) for the FB algorithm (see also lines 6-11 of Algorithm AI.1), and in (I.15) and (I.16) for the FFBS algorithm (see also lines 3-7 Algorithm AI.3).

In addition, as presented in Figure AI.3, the filtered state densities (Eqs. (I.7) for FB, and (I.15) and (I.16) for FFBS) computed at each time step of the forward pass, must be loaded into the memory in order to compute smoothed state densities (Eqs. (II.5) and (II.6)) in the backward pass. The bottleneck in terms of memory storage, for both algorithms, is therefore the size of the matrix of $N \times T$ floating-points values.

A memory complexity of $\mathcal{O}(N^2T)$ is often attributed to FB and FFBS algorithms in the literature. In such analysis, memory is assigned to $\xi_{t|T}(i,j)$ which requires T matrices of $N \times N$ floating points. However, as shown in the transition update Equation (II.8), only the summation over time is required; hence only one matrix of N^2 floating points is required for storing $\sum_{t=1}^{T} \xi_{t|T}(i,j)$. In addition, the N floating point values required for storing the filtered state estimate, $\gamma_{t|t}$, (i.e., N^2+N), are also unavoidable.

Table AI.1 details an analysis of the worst-case time and memory complexity for the FB and FFBS algorithms processing an observation sequence $o_{1:T}$ of length T. Time complexity represents the worst-case number of operations required for one iteration of BW. A BW iteration involves computing one forward and one backward pass with $o_{1:T}$. Memory complexity is the worst-case number of 32 bit words needed to store the required temporary variables in RAM. Analysis shows that the FFBS algorithm requires slightly fewer computations ($3N^2T+NT$ multiplications, $2NT$ divisions and $3NT$ additions) than the FB algorithm. It can be also seen that both algorithms require the same amount of storage: an array of $NT+N^2+N$ floating point values.

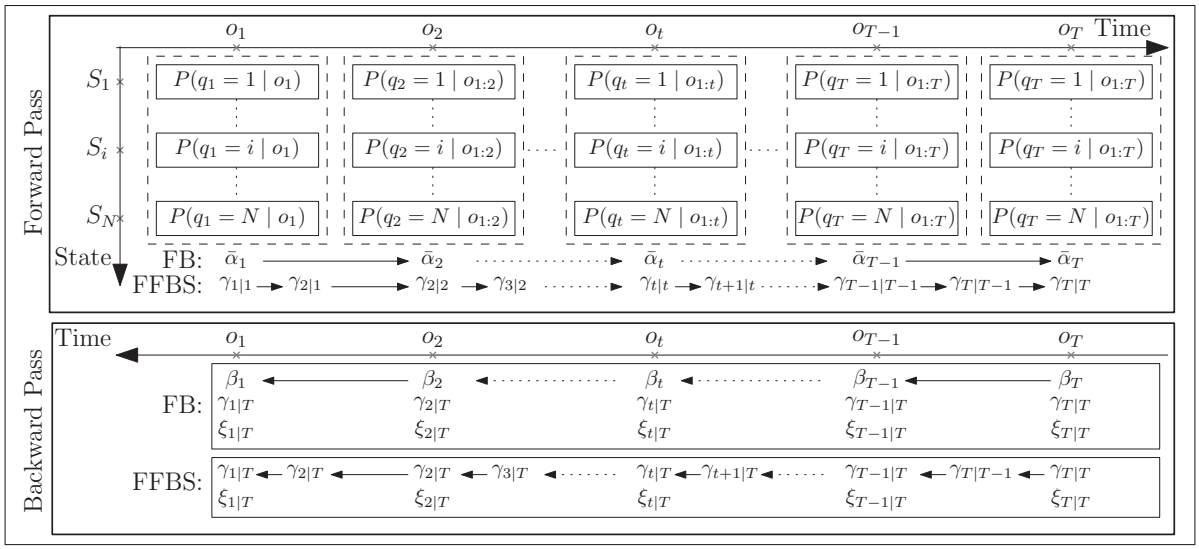


Figure AI.3 An illustration of the values that require storage during the forward and backward passes of the FB and FFBS algorithms. During the forward pass the filtered state densities at each time step ($N \times T$ floating-points values) must be stored into the internal system memory, to be then used to compute the smoothed state densities during the backward pass

Table AI.1 Worst-case time and memory complexity analysis for the FB and FFBS for processing an observation sequence $o_{1:T}$ of length T with an N state HMM. The scaling procedure as described in Rabiner (1989) and shown in Algorithms AI.1 and AI.2 is taken into consideration with the FB algorithm

Computations	**Time**			**Memory**
	# Multiplications	# Divisions	# Additions	
	Forward-Backward (FB)			
α_t	$N^2T+NT-N^2$	$NT-N$	N^2T-N^2+N	NT
β_t	$2N^2T-2N^2$	NT	$N^2T-NT-N^2+N$	–
$\sum_t^T \gamma_{t\|T}$	NT	NT	$NT-T$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$3N^2T-3N^2$	N^2T-N^2	$N^2T-NT-N^2+N$	N^2
Total	$6N^2T+2NT-6N^2$	$N^2T+3NT-N^2-N$	$3N^2T+NT-3N^2+3N-T$	$NT+N^2+N$
	Forward Filtering Backward Smoothing (FFBS)			
$\gamma_{t\|t}$	$N^2T+NT-N^2$	NT	N^2T-N^2+N	NT
β_t	–	–	$N^2T-NT-N^2+N$	–
$\sum_t^T \gamma_{t\|T}$	–	–	$NT-T$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$2N^2T-2N^2$	N^2T-N^2	$N^2T-NT-N^2+N$	N^2
Total	$3N^2T+NT-3N^2$	$N^2T+NT-N^2$	$3N^2T+NT-3N^2+3N-T$	$NT+N^2+N$

With both FB and FFBS algorithms a memory problem stems from its linear scaling with the length of the observation sequence T. Applications that require estimating HMM

parameters from a long sequence will require long training times and large amounts of memory, which may be prohibitive for the training process.

As described next, some alternatives for computation of exact smoothed density have been proposed to alleviate this issue. However, these alternatives are either still partially dependent on the sequence length T, or increase the computational time by orders of magnitude to achieve a constant memory complexity.

I.3.2 Checkpointing Algorithm

The checkpointing algorithm stores only some reference columns or checkpoints, instead of storing the whole matrix of filtered state densities (Grice et al., 1997; Tarnas and Hughey, 1998). Therefore, it divides the input sequence into $\sqrt{T}$ sub-sequences. While performing the forward pass, it only stores the first column of the forward variables for each sub-sequences. Then, during the backward pass, the forward values for each sub-sequence are sequentially recomputed, beginning with their corresponding checkpoints. Its implementation reduces the memory complexity to $\mathcal{O}(N\sqrt{T})$, yet increases the time complexity to $\mathcal{O}(N^2(2T - \sqrt{T}))$. The memory requirement can be therefore traded-off against the time complexity.

I.3.3 Forward Only Algorithm

A direct propagation of the smoothed densities in forward-only manner is an alternative that has been proposed in the communication field (Elliott et al., 1995; Narciso, 1993; Sivaprakasam and Shanmugan, 1995) and re-discovered in bioinformatics (Churbanov and Winters-Hilt, 2008; Miklos and Meyer, 2005). The basic idea is to directly propagate all *smoothed* information in the forward pass

$$\sigma_t(i,j,k) \triangleq \sum_{\tau=1}^{t-1} P(q_\tau = i, q_{\tau+1} = j, q_t = k \mid o_{1:t}, \lambda),$$

which represents the probability of having made a transition from state S_i to state S_j at some point in the past ($\tau < t$) and of ending up in state S_k at the current time t. At the next time step, $\sigma_{t+1}(i,j,k)$ can be recursively computed from $\sigma_t(i,j,k)$ using:

$$\sigma_t(i,j,k) = \sum_{n=1}^{N} \sigma_{t-1}(i,j,n) a_{nk} b_k(o_t) + \alpha_{t-1}(i) a_{ik} b_k(o_t) \delta_{jk} \tag{I.19}$$

from which the smoothed state densities can be obtained, at time T, by marginalizing over the states.

By eliminating the backward pass, the memory complexity becomes $\mathcal{O}(N^2 + N)$, which is independent T. However, it is achieved at the expenses of increasing the time complexity to $\mathcal{O}(N^4 T)$ as can be seen in the four-dimensional recursion (I.19). Due to this computational overhead for training, this forward-only approach has not gained much popularity in practical applications.

I.4 Efficient Forward Filtering Backward Smoothing (EFFBS) Algorithm

In this section a novel and efficient alternative that is able to compute the exact smoothed densities with a memory complexity that is independent of T, and without increasing time complexity is described. The proposed alternative – termed the Efficient Forward Filtering Backward Smoothing (EFFBS) – is an extension of the FFBS algorithm. It exploits the facts that a backward pass of the FFBS algorithm does not require any access to the observations and that the smoothed state densities are recursively computed from each other in a backward pass.

Instead of storing all the predictive and filtered state densities (see Eqs. (I.13) and (I.14)) for each time step $t = 1, \ldots, T$, the EFFBS recursively recomputes their values starting from the end of the sequence. Accordingly, the memory complexity of the algorithm becomes independent of the sequence length. That is, by storing only the last values of $\gamma_{T|T}$ and $\gamma_{T|T-1}$ from the forward pass, the previously-computed filtered and predictive

densities can be recomputed recursively, in parallel with the smoothed densities (refer to Algorithm AI.5).

Let the notations $\odot$ and $(\div)$ be the term-by-term multiplication and division of two vectors, respectively. Eqs. (I.14) and (I.16) can be written as:

$$\gamma_{t|t} = \frac{\left[\gamma_{t|t-1} \odot b_t\right]}{\left\|\gamma_{t|t}\right\|_1} \tag{I.20}$$

$$\gamma_{t|t-1} = A'\gamma_{t-1|t-1} \tag{I.21}$$

where $b_t = [b_1(o_t), \ldots b_N(o_t)]'$, $\|.\|_1$ is the usual 1-norm of a vector (column sum) and the *prime* superscript denotes matrix transpose. The backward pass of the EFFBS algorithm starts by using only the column-vector filtered state estimate of the last observation, $\gamma_{T|T}$, resulting from the forward pass. Then, for $t = T-1, \ldots 1$, $\gamma_{t|t-1}$ is recomputed from $\gamma_{t|t}$ using:

$$\gamma_{t|t-1} = \frac{\left[\gamma_{t|t}(\div)b_t\right]}{\left\|\gamma_{t|t-1}\right\|_1} \tag{I.22}$$

from which, the filtered state estimate can be recomputed by

$$\gamma_{t-1|t-1} = \left[A'\right]^{-1}\gamma_{t|t-1} \tag{I.23}$$

This step requires the inverse of the transposed transition matrix $[A']^{-1}$ to solve N linear equations with N unknowns. The only necessary condition is the nonsingularity of the transition matrix A to be invertible. This is a weak assumption that has been used in most of the studies that addressed the statistical properties of HMMs (c.f. Ephraim and Merhav, 2002). In particular, it holds true in applications that use fully connected HMMs, where the elements of the transition matrix are positives (i.e., irreducible and aperiodic).

I.4.1 Complexity Analysis of the EFFBS Algorithm

Algorithm AI.5: EFFBS($o_{1:T}, \lambda$): Efficient Forward Filtering Backward Smoothing

Output: $\gamma_{t|T}(i) = P(q_t = i \mid o_{1:T})$ and $\xi_{t|T}(i,j) = P(q_t = i, q_{t+1} = j \mid o_{1:T})$

`// Forward Filtering: the filtered,` $\gamma_{t|t}$`, and predictive,` $\gamma_{t+1|t}$`, state densities are column vectors of length` N`. Their values are recursively computed, however they are not stored at each time step,` t`, but recomputed in the backward pass to reduce the memory complexity.`

for $i = 1, \ldots, N$ **do** `// Initialization`
  $\gamma_{1|0}(i) = \pi_i$
for $t = 1, \ldots T-1$ **do** `// Induction`
  for $i = 1, \ldots, N$ **do**
    $\gamma_{t|t}(i) \leftarrow \frac{\gamma_{t|t-1}(i) b_i(o_t)}{\sum_{k=1}^{N} \gamma_{t|t-1}(k) b_k(o_t)}$
  for $j = 1, \ldots, N$ **do**
    $\gamma_{t+1|t}(j) \leftarrow \sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}$
for $t = 1, \ldots T$ **do** `// Evaluation`
  $\ell_T(\lambda) \leftarrow \ell_T(\lambda) + \log \sum_{i=1}^{N} b_i(o_t) \gamma_{t|t-1}(i)$

`// Backward Smoothing: the last values of the filtered,` $\gamma_{T|T}$`, and predictive,` $\gamma_{T|T-1}$`, state densities are now used to recompute their previous values.`

$A = a^{-1}$ `// matrix inversion only once per iteration`
for $t = T-1, \ldots, 1$ **do** `// Induction`
  for $i = 1, \ldots, N$ **do**
    $\gamma_{t|t}(i) = \sum_{k=1}^{N} A_{ki} \gamma_{t+1|t}(k)$
    $\gamma_{t|t-1}(i) = \frac{\gamma_{t|t}(i)/b_i(o_t)}{\sum_{k=1}^{N} \gamma_{t|t}(k)/b_k(o_t)}$
    for $j = 1, \ldots, N$ **do**
      $\xi_{t|T}(i,j) \leftarrow \frac{\gamma_{t|t}(i) a_{ij}}{\sum_{i=1}^{N} \gamma_{t|t}(i) a_{ij}} \gamma_{t+1|T}(j)$
    $\gamma_{t|T}(i) \leftarrow \sum_{j=1}^{N} \xi_{t|T}(i,j)$

In order to compute the smoothed densities in a constant memory complexity, the EFFBS algorithm must recompute $\gamma_{t|t}$ and $\gamma_{t|t-1}$ values in its backward pass, instead of storing them, which requires twice the effort of the forward pass computation. This recomputation however takes about the same amount of time required for computing the β values in the backward pass of the FB algorithm.

Table AI.2 presents a worst-case analysis of time and memory complexity for the EFFBS. The proposed algorithm is comparable to both FB and FFBS algorithms (see Table AI.1) in terms of computational time, with an overhead of about $N^2T + NT$ multiplications

Table AI.2 Worst-case Time and Memory Complexity of the EFFBS Algorithm for Processing an Observation Sequence $o_{1:T}$ of Length T with an N State Ergodic HMM

Computation	**Time**			**Memory**
	# Multiplications	# Divisions	# Additions	
$\gamma_{T\|T}$	$N^2T+NT-N^2$	NT	N^2T-N^2+N	N
$\gamma_{t\|t}$ and $\gamma_{t\|t-1}$	$N^2T+NT-N^2$	NT	N^2T-N^2+N	$2N$
A^{-1}	$\frac{N^3}{3}+\frac{N^2}{2}-\frac{N}{3}$	–	$\frac{N^3}{3}+\frac{N^2}{2}-\frac{5N}{6}$	N^2
$\sum_t^T \gamma_{t\|T}$	–	–	$N^2T-NT-N^2+N$	N
$\sum_{t=1}^{T-1} \xi_{t\|T}$	$2N^2T-2N^2$	N^2T-N^2	$N^2T-NT-N^2+N$	N^2
Total	$4N^2T+2NT+$ $\frac{N^3}{3}-\frac{7N^2}{2}-\frac{N}{2}$	$N^2T+2NT-N^2$	$4N^2T-2NT+$ $\frac{N^3}{3}-\frac{7N^2}{2}+\frac{19N}{6}$	N^2+5N

Table AI.3 Time and Memory Complexity of the EFFBS Versus Other Approaches (Processing an Observation Sequence $o_{1:T}$ of Length T, for an Ergodic HMM with N States)

Algorithm	**Time**	**Memory**
FB	$\mathcal{O}(N^2T)$	$\mathcal{O}(NT)$
FFBS	$\mathcal{O}(N^2T)$	$\mathcal{O}(NT)$
Checkpointing	$\mathcal{O}(N^2(2T-\sqrt{T}))$	$\mathcal{O}(N\sqrt{T})$
Forward-Only	$\mathcal{O}(N^4T)$	$\mathcal{O}(N)$
EFFBS	$\mathcal{O}(N^2T)$	$\mathcal{O}(N)$

and additions. An additional computational time is required for the inversion of the transition matrix, *which is computed only once for each iteration.* The classical Gauss-Jordan elimination, may be used for instance to achieve this task in $\mathcal{O}(N^3)$. Faster algorithms for matrix inversion could be also used. For instance, inversion based on Coppersmith–Winograd algorithm (Coppersmith and Winograd, 1990) is $\mathcal{O}(N^{2.376})$.

The EFFBS only requires two column vectors of N floating point values on top of the unavoidable memory requirements (N^2+N) of the smoothed densities. In addition, a matrix of N^2 floating point values is required for storing the inverse of the transposed transition matrix $[A']^{-1}$. Table AI.3 compares the time and memory complexities of existing fixed-interval smoothing algorithms for computing the state densities to the proposed EFFBS algorithm.

I.4.2 Case Study in Host-Based Anomaly Detection

Intrusion Detection Systems (IDSs) are used to identify, assess, and report unauthorized computer or network activities. Host-based IDSs (HIDSs) are designed to monitor the host system activities and state, while network-based IDSs monitor network traffic for multiple hosts. In either case, IDSs have been designed to perform misuse detection – looking for events that match patterns corresponding to known attacks – and anomaly detection – detecting significant deviations from normal system behavior.

Operating system events are usually monitored in HIDSs for anomaly detection. Since system calls are the gateway between user and kernel mode, early host-based anomaly detection systems monitor deviation in system call sequences. Various detection techniques have been proposed to learn the normal process behavior through system call sequences (Warrender et al., 1999). Among these, techniques based on discrete Hidden Markov Models (HMMs) have been shown to shown to provide high level of performance (Warrender et al., 1999).

The primary advantage of anomaly-based IDS is the ability to detect novel attacks for which the signatures have not yet been extracted. However, anomaly detectors will typically generate false alarms mainly due to incomplete data for training. A crucial step to design an anomaly-based IDS is to acquire sufficient amount of data for training. Therefore, a large stream of system call data is usually collected from a process under normal operation in a secured environment. This data is then provided for training the parameters of an ergodic HMM to fit the normal behavior of the monitored process.

A well trained HMM should be able to capture the underlying structure of the monitored application using the temporal order of system calls generated by the process. Once trained, a normal sequence presented to HMM should produce a higher likelihood value than for a sequence that does not belong to the normal process pattern or language. According to a decision threshold the HMM-based Anomaly Detection System (ADS) is able to discriminate between the normal and anomalous system call sequences.

The sufficient amount of training data depends on the complexity of the process and is very difficult to be quantified *a priori*. Therefore, the larger the sequence of system calls provided for training, the better the discrimination capabilities of the HMM-based ADS and the fewer the rate of false alarms. In practice however, the memory complexity of the FB (or FFBS) algorithm could be be prohibitively costly when learning is performed for a large sequence of system calls, as previously described. In their experimental work on training HMM for anomaly detection using large sequences of system calls, Warrender et al. (1999) state:

> On our largest data set, HMM training took approximately two months, while the other methods took a few hours each. For all but the smallest data sets, HMM training times were measured in days, as compared to minutes for the other methods.

This is because the required memory complexity for training the HMM with the FB algorithm exceeds the resources available, causing overflows from internal system memory, e.g., Random-Access Memory (RAM), to disk storage. Accessing paged memory data on a typical disk drive is approximately one million time slower than accessing data in the RAM.

For example, if an ergodic HMM with $N = 50$ states must be trained using an observation sequence of length $T = 100 \times 10^6$ system calls collected from a complex process. The time complexity per iteration of the FFBS algorithm (see Table AI.1):

$$Time_{\{FFBS\}} = 75.5 \times 10^{10} \text{ MUL} + 25.5 \times 10^{10} \text{ DIV} + 75.5 \times 10^{10} \text{ ADD}$$

is comparable to that of the EFFBS algorithm (see Table AI.2):

$$Time_{\{EFFBS\}} = 101 \times 10^{10} \text{ MUL} + 26 \times 10^{10} \text{ DIV} + 99 \times 10^{10} \text{ ADD}$$

However, the amount of memory required by the EFFBS algorithm is 0.01 MB, which is negligible compared to the 20,000 MB (or 20 GB) required by the FFBS algorithm. Therefore, the EFFBS algorithm could form a practical approach for designing low-footprint intrusion detection systems.

I.5 Conclusion

This paper considered the memory problem of the Forward-Backward algorithm when trained with a long observation sequence. This is an interesting problem facing various HMM applications - ranging from computer security to robotic navigation systems and bioinformatics. To alleviate this issue, an Efficient Forward Filtering Backward Smoothing (EFFBS) algorithm is proposed to render the memory complexity independent of the sequence length, without incurring any considerable computational overhead and without any approximations. A detailed complexity analysis has shown that the proposed algorithm is more efficient than existing solutions in terms of both memory and computational requirements.

APPENDIX II

A COMPARISON OF TECHNIQUES FOR ON-LINE INCREMENTAL LEARNING OF HMM PARAMETERS IN ANOMALY DETECTION*

Abstract

Hidden Markov Models (HMMs) have been shown to provide a high level performance for detecting anomalies in intrusion detection systems. Since incomplete training data is always employed in practice, and environments being monitored are susceptible to changes, a system for anomaly detection should update its HMM parameters in response to new training data from the environment. Several techniques have been proposed in literature for on-line learning of HMM parameters. However, the theoretical convergence of these algorithms is based on an infinite stream of data for optimal performances. When learning sequences with a finite length, incremental versions of these algorithms can improve discrimination by allowing for convergence over several training iterations. In this paper, the performance of these techniques is compared for learning new sequences of training data in host-based intrusion detection. The discrimination of HMMs trained with different techniques is assessed from data corresponding to sequences of system calls to the operating system kernel. In addition, the resource requirements are assessed through an analysis of time and memory complexity. Results suggest that the techniques for incremental learning of HMM parameters can provide a higher level of discrimination than those for on-line learning, yet require significantly fewer resources than with batch training. Incremental learning techniques may provide a promising solution for adaptive intrusion detection systems.

*This chapter is published in proceedings of the second IEEE international conference on Computational Intelligence for Security and Defense Applications (CISDA09)

II.1 Introduction

Intrusion Detection Systems (IDSs) are used to identify, assess, and report unauthorized computer or network activities. Host-based IDSs (HIDSs) are designed to monitor the host system activities and state, while network-based IDSs monitor network traffic for multiple hosts. In either case, IDSs have been designed to perform misuse detection – looking for events that match patterns corresponding to known attacks – and anomaly detection – detecting significant deviations from normal system behavior.

Operating system events are usually monitored in HIDSs for anomaly detection. Since system calls are the gateway between user and kernel mode, early host-based anomaly detection systems monitor deviation in system call sequences (Forrest et al., 1996). Various detection techniques have been proposed to learn the normal process behavior through system call sequences (Warrender et al., 1999). Among these, techniques based on discrete Hidden Markov Models (HMMs) have been shown to shown to provide high level of performance (Warrender et al., 1999).

HMM is stochastic process for sequential data (Rabiner, 1989). Given an adequate amount of system call training data, HMM-based anomaly detectors can efficiently model the normal process behavior. A well trained HMM should be able to capture the underlying structure of the monitored application using the temporal order of system calls generated by the process. Once trained, an HMM provides a compact model, with tolerance to noise and uncertainty, which allows a fast evaluation during operations[1]. A normal sequence presented to HMM should produce a higher likelihood value than for a sequence that does not belong to the normal process pattern or language. Their ability to discriminate between normal and malicious sequences have been discussed in literature (Gao et al., 2002; Hoang et al., 2003; Wang et al., 2004). The effects on performance of

[1]In contrast, matching techniques that are based on look-up tables, e.g., STIDE (sequence time-delay embedding) must compare inputs to all normal training sequences. The number of comparisons increases exponentially with the detector window size DW, while for HMM evaluation the time complexity grows linearly with DW.

the training set size, irregularity of the process, anomaly types, and number of hidden states of HMM were recently investigated in Khreich et al. (2009b).

The primary advantage of anomaly-based IDS is the ability to detect novel attacks for which the signatures have not yet been extracted. However, anomaly detectors will typically generate false alarms mainly due to incomplete data for training, poor modeling, and difficulty in obtaining representative labeled data for validation. In practice, it is very difficult to acquire (collect and label) comprehensive data sets to design a HIDS for anomaly detection. Therefore, a major requirement for an anomaly detection system (ADS) is the ability to accommodate new data without the need to restart the training process with all accumulated data.

Most research found in literature for HMM-based anomaly detection using system calls assume being provided with a sufficient amount of data. Furthermore, the monitored process is not static – changes in the environment may occur, such as application update. This is also the case when fine tuning a base model to a specific host platform. Therefore, HMM parameters should be refined incrementally over time by accommodating newly acquired training data, to better fit the normal process behavior.

Standard techniques for training HMM parameters involve batch learning, based either on the Baum-Welch (BW) algorithm (Baum et al., 1970), a specialized expectation maximization (EM) technique (Dempster et al., 1977), or on numerical optimization methods, such as the Gradient Descent (GD) algorithm (Levinson et al., 1983). Both approaches are iterative algorithms for maximizing the likelihood estimate (MLE) of the data. For a batch learning technique, the data sequence is assumed to be finite. Each training iteration of BW or GD involves observing all subsequences in the presented block[2] for training prior to updating HMM parameters. Successive iterations continues until some stopping criterion is achieved (e.g., likelihood drop on a validation set). Given a new

[2]A block of data is defined as a sequence of system call observations that has been segmented into overlapping subsequences according to a user-defined window size.

block of data, an HMM trained with BW or GD must be trained from start using all cumulative training data.

As an alternative, incremental algorithm updates the HMM parameters after each subsequence, yet is allowed to perform several iterations over all subsequences within the block (refer to Figure AII.1). Some desirable characteristics for incremental learning include the ability to update HMM parameters from new training data, without requiring access to the previously-learned training data and without corrupting previously acquired knowledge (Polikar et al., 2001).

In contrast, for an on-line learning technique, the data sequence is assumed to be infinite. Such techniques update HMM parameters after observing each subsequence, with no iterations. User-defined hyper-parameters remain constant or they are allowed to degrade monotonically over time. Given a new block, an HMM that performs on-line learning continues the training seamlessly. In practice when learning sequences with finite length, on-line learning may lead to poor performance.

Several techniques have been proposed in literature for different real-world applications. Among these, on-line learning techniques are based on the current sequence of observations for optimizing the objective function (commonly the MLE), and updating the HMM parameters. As with batch learning, they can also be divided into EM-based (Mizuno et al., 2000) and gradient-based (Baldi and Chauvin, 1994; Ryden, 1998; Singer and Warmuth, 1996) learning techniques. These on-line techniques are extended in this work to incremental learning by allowing them to iterate over each block of data and by resetting the learning rates when a new block is presented. Another solution consists of learning an HMM for each new block of data then merging it with old ones using weight-averaging (Hoang and Hu, 2004).

The objectives of this paper are to compare the techniques for on-line and incremental learning of HMMs parameters when applied for anomaly HIDS application using system calls sequences. A synthetic generator of normal data plus injection of anomaly has been

utilized to avoid various drawbacks encountered when experimenting with real data. The receiver operating characteristics (ROC) curves and the area under the ROC curve (AUC) are used as a measure of performance (Fawcett, 2006). Analytical comparison of convergence time and resources requirements are also provided and discussed.

The rest of this paper is organized as follows. The next section discusses the importance of incremental update for anomaly detection. Section II.3 presents techniques for batch, on-line and incremental learning for HMM parameters. The experimental methodology in II.4 describes data generation, evaluation methods and performance metrics. Finally, simulation results are discussed in II.5.

II.2 Incremental Learning in Anomaly Detection System

An crucial step to design an ADS is to acquire sufficient amount of data for training. In practice however, it is very difficult to collect, analyze and label comprehensive data sets due to many reasons that range from technical to ethical. In addition, even in the simplest scenario where no change in the environment is assumed, characterizing the sufficient amount of data required for building an efficient ADS is not a trivial task. In practice, limited data is always provided for training, and the computer environment is always susceptible to dynamic changes. This work focuses on providing solutions to the limited data problem as described with the following practical scenarios.

Given some amount of normal system call data for training, an ADS based on HMM could be trained, optimized and validated off-line to provide some acceptable performance in terms of false and true positive rates. However, during operations, monitoring a centralized server for instance, the system is susceptible to produce a higher rate of false alarms than expected and tolerated by the system administrator. This is largely due to the limited data that is the provided for training. The anomaly detector will have a limited view of the normal process behavior, and rare events will be mistakenly considered as anomalous. Accordingly, the HMM detector should be refined, i.e., trained on some

additional normal data when it becomes available, to better fit the normal behavior of process in consideration.

As a part of the detection system, the system administrator plays an important role for providing such new data. When an alarm is raised the suspicious system call subsequences are logged and the system administrator starts an investigation into other evidence of an attack. If an intrusion attempt is detected the response team will act to limit the damage, and the forensic analysis team try to find the cause of the successful attack. Otherwise, it is considered as a false alarm and the logged subsequences (which are possibly rare events) are tagged as normal and collected for updating the HMM detector. One challenge is the efficient integration of this newly-acquired data into the ADS without corrupting the existing knowledge structure, and thereby degrading the performance.

II.3 Techniques for Learning HMM Parameters

A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms. A latent Markov chain having comprising N states in the finite-state space $S = \{S_1, S_2, ..., S_N\}$, and a set of observation discrete probability distributions $b_j(v)$, each one associated with a state (Ephraim and Merhav, 2002; Rabiner, 1989). Starting from an initial state S_i, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij} $(1 \leq i, j \leq N)$. The process then emits a symbol v according to the output probability distribution $b_j(v)$ of the current state S_j. The model is therefore parametrized by the set $\lambda = (\pi, A, B)$, where vector $\pi = \{\pi_i\}$ is initial state probability distribution, matrix $A = \{a_{ij}\}$ denotes the state transition probability distribution, and matrix $B = \{b_j(k)\}$ is the state output probability distribution. Both A, B, and π' are row stochastic, which impose the following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \forall i, \sum_{k=1}^{M} b_j(k) = 1 \forall j, \text{ and } \sum_{i=1}^{N} \pi_i = 1 \tag{II.1}$$

$$a_{ij}, b_j(k), \text{ and } \pi_i \in [0,1], \forall ijk \tag{II.2}$$

II.3.1 Batch Learning

The target in HMM parameters learning is to train the model λ to best fit the observed batch of data $o_{1:T}$. The estimation of HMM parameters is frequently performed according to the maximum likelihood estimation (MLE) criterion[3]. MLE consists of maximizing the log-likelihood

$$\ell_T(\lambda) \triangleq \log \Pr(o_{1:T} \mid \lambda) \tag{II.3}$$

of the training data over HMM parameters space (Λ):

$$\lambda^* = \text{argmax}_{\lambda \in \Lambda} \ell_T(\lambda) \tag{II.4}$$

Unfortunately, since the log-likelihood depends on missing information (the latent states), there is no known analytical solution to the training problem. In practice, iterative optimization techniques (briefly described below) such as the Baum-Welch algorithm, a special case of the Expectation-Maximization (EM), or alternatively the standard numerical optimization methods such as gradient descent are usually used for this task.

In either case, the optimization requires the evaluation of the log-likelihood value (II.3) at each iteration, and the estimation of the conditional state densities. That is, the *smoothed a posteriori* conditional state density:

$$\gamma_t(i) \triangleq \Pr(q_t = i \mid o_{1:T}, \lambda) \tag{II.5}$$

and the *smoothed a posteriori* conditional joint state density:

$$\xi_t(i,j) \triangleq \Pr(q_t = i, q_{t+1} = j \mid o_{1:T}, \lambda) \tag{II.6}$$

[3]Other criteria such as the maximum mutual information (MMI), and minimum discrimination information (MDI) could be also used for estimating HMM parameters. However, the widespread usage of the MLE for HMM is due to its attractive statistical properties – consistency and asymptotic normality – proved under quite general conditions.

The Forward-Backward (FB) (Rabiner, 1989) or the numerically more stable Forward-Filtering Backward-Smoothing (FFBS) (Ephraim and Merhav, 2002) algorithms are typically used for computing the log-likelihood value (II.3) and the smoothed state densities of Eqs. (II.5) and (II.6).

The Baum-Welch (BW) algorithm (Baum et al., 1970) is an Expectation-Maximization (EM) algorithm (Dempster et al., 1977) specialized for estimating HMM parameters. Instead of a direct maximization of the log-likelihood (II.3) BW optimizes the auxiliary $\mathcal{Q}$-function:

$$\mathcal{Q}_T(\lambda, \lambda^{(k)}) = \sum_{q \in S} \Pr(o_{1:T}, q_{1:T} \mid \lambda^{(k)}) \log \Pr(o_{1:T}, q_{1:T} \mid \lambda) \tag{II.7}$$

which is the expected value of the complete-data log-likelihood and hence easier to be optimized. This is done by alternating between the expectation step (E-step) and maximization step (M-step). The E-step uses the FB or FFBS algorithms to compute the state densities (Eqs. II.5 and II.6) which are then used, in the M-step to re-estimate the model parameters:

$$\begin{aligned} \pi_i^{(k+1)} &= \gamma_1^{(k)}(i) \\ a_{ij}^{(k+1)} &= \frac{\sum_{t=1}^{T-1} \xi_t^{(k)}(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ b_j^{(k+1)}(m) &= \frac{\sum_{t=1}^{T} \gamma_t^{(k)}(j) \delta_{o_t v_m}}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned} \tag{II.8}$$

The Kronecker delta δ_{ij} is equal to one if $i = j$ and zero otherwise. Starting with an initial guess of the HMM parameters, λ^0, each iteration k, of the E- and M-step is guaranteed to increase the likelihood of the observations giving the new model until a convergence to a stationary point of the likelihood is reached (Baum et al., 1970).

In contrast, standard numerical optimization methods work directly with the log-likelihood function (II.3) and its derivatives. Starting with an initial guess of HMM parameters λ^0,

the gradient descent (GD) updates the model at each iteration k using:

$$\lambda^{(k+1)} = \lambda^{(k)} + \eta_k \nabla_\lambda \ell_T(\lambda^{(k)}) \tag{II.9}$$

where the learning rate η_k could be fixed, or adjusted at each iteration. One way of computing the gradient of the likelihood $\nabla_\lambda \ell_T(\lambda^{(k)})$ is by using the values of the conditional densities (Eqs. II.5 and II.6) obtained from the FB or FFBS algorithms:

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{a_{ij}} \tag{II.10}$$

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial b_j(m)} = \frac{\sum_{t=1}^{T} \gamma_t(j) \delta_{o_t v_m}}{b_j(m)} \tag{II.11}$$

However, with the numerical optimization methods HMM parameters are not guaranteed to stay within their space limits. As described next, the parameters constraints (Eqs. II.1 and II.2) must therefore be imposed explicitly through a re-parametrization, to reduce the problem to unconstrained optimization.

Since, at each iteration, both BW and GD algorithms rely on the fixed-interval smoothing algorithms (FB or FFBS) to compute the state conditional densities, they are therefore performing a batch learning approach. This is because these fixed-interval smoothing algorithms require an access to the end of sequence in order to compute the smoothed densities. Similarly, when learning from a block of multiple subsequences, each iteration of the BW and GD requires the averaged smoothed densities over all the subsequences in the block. Therefore, the block must have a finite number of subsequences, and all the subsequences are visited at each iteration. Consequently, when provided with a new block of subsequences the training process must be restarted using the accumulated (old and new) data to accommodate the new data.

II.3.2 On-line and Incremental Learning

Figure AII.1 presents an illustration of the batch, on-line, and incremental learning approaches when provided with subsequent blocks each comprises R observation subsequences. When the first block (D_1) is presented, all algorithms start with the same initial guess of HMM parameters (λ_0). At each iteration k, batch algorithms update the model parameters using the averaged state densities (Eqs. II.5 and II.6) over all the subsequences in the block until stopping criteria are met, then the first operational model (λ_1) is produced. On-line algorithms directly update the model parameters using the stated densities based on each subsequence and output λ_1 upon reaching the last subsequence in the block. This constitutes one iteration of the incremental algorithms which then re-iterate until reaching the stopping criteria before producing λ_1.

When the second block (D_2) is presented, batch algorithms restart the training procedure, using all accumulated training data ($D_1 \cup D_2$). The on-line algorithms however resumes training from the previous model (λ_1) using only the current block (D_2) without iterations, while the incremental algorithms will re-iterate on D_2 until the stopping criteria are met.

On-line learning techniques for HMM parameters can be broadly divided into EM-based (Mizuno et al., 2000) or gradient-based (Baldi and Chauvin, 1994; Ryden, 1998; Singer and Warmuth, 1996) optimization of the log-likelihood function. These techniques are essentially derived from their batch counterparts, however the key difference is that the optimization and the HMM update are based on the currently presented subsequence of observations without iterations. EM-based techniques employ an indirect maximization of log-likelihood, through the complete log-likelihood (II.7), in which the E-step is performed on each subsequence of observation and then the model parameters are updated (Mizuno et al., 2000). Gradient-based techniques however directly maximize the log-likelihood function (II.3) and then update HMM parameters after processing each subsequence of observation. Several gradient-based optimization algorithms have been proposed, such as the GD algorithm (Baldi and Chauvin, 1994), the exponentiated

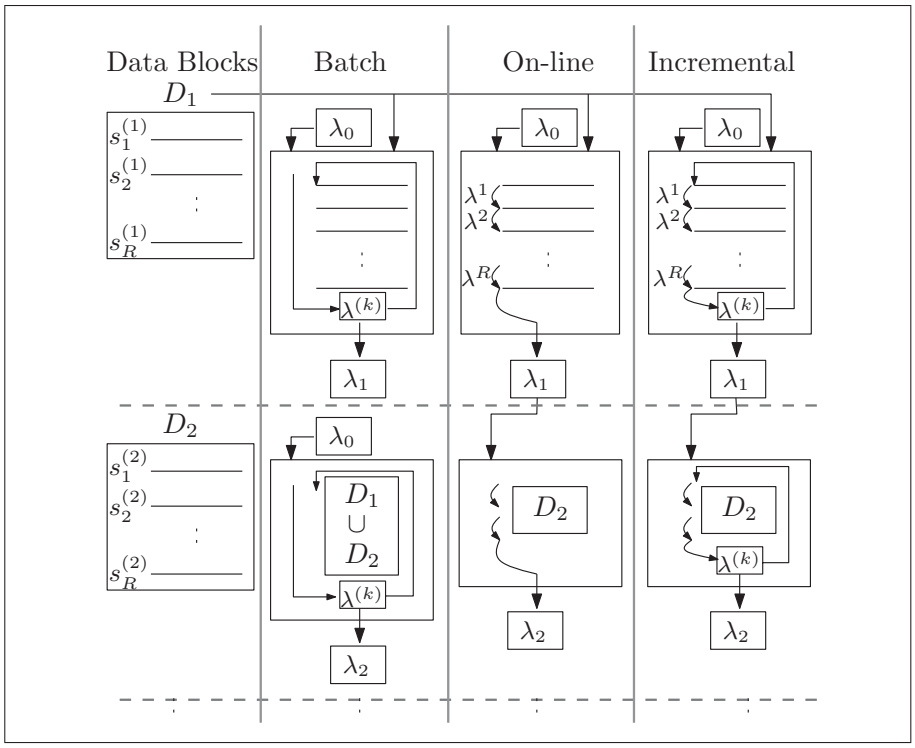


Figure AII.1 Illustration of batch, on-line, and incremental learning approaches when learning from subsequent blocks $(D_1, D_2, \ldots)$ of R observation subsequences, provided at different time intervals

gradient framework which also minimizes the model parameters divergence (Singer and Warmuth, 1996), and the recursive estimation technique (Ryden, 1998).

The on-line learning technique proposed by Mizuno et al. (2000) is based on the BW algorithm, however applies a decayed accumulation of the state densities and a direct update of the model parameters after each subsequence of observations (Mizuno et al., 2000). Starting with an initial model λ_0, the conditional state densities are recursively computed after processing each subsequence (r) of observations of length T by:

$$\sum_{t=1}^{T-1} \xi_t^{r+1}(i,j) = (1-\eta_r) \sum_{t=1}^{T-1} \xi_t^r(i,j) + \eta_r \sum_{t=1}^{T-1} \xi_t^{r+1}(i,j) \tag{II.12}$$

$$\sum_{t=1}^{T} \gamma_t^{r+1}(j)\delta_{o_t k} = (1-\eta_r) \sum_{t=1}^{T} \gamma_t^r(j)\delta_{o_t k} + \eta_r \sum_{t=1}^{T} \gamma_t^{r+1}(j)\delta_{o_t k} \tag{II.13}$$

and the model parameters are then directly updated using (II.8). The learning rate η_k is proposed in polynomial form $\eta_r = c(\frac{1}{r})^d$ for some positive constants c and d.

Based on the GD (II.9) of the negative likelihood, the on-line algorithm for HMM parameters estimations introduced by Baldi and Chauvin (1994) employs a softmax parametrization to transform the constraint optimization into an unconstrained one. This is achieved by mapping the bounded space (a, b) to the unbounded space (u, v):

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_k e^{u_{ik}}} \text{ and } b_j(k) = \frac{e^{v_j(k)}}{\sum_z e^{v_j(z)}} \tag{II.14}$$

The transformed parameters are then updated, after each subsequence of observations, as follows:

$$u_{ij}^{r+1} = u_{ij}^r + \eta \sum_{t=1}^{T} \Big(\xi_t^{r+1}(i,j) - a_{ij} \gamma_t^{r+1}(i) \Big) \tag{II.15}$$

$$v_j^{r+1}(k) = v_j^r(k) + \eta \sum_{t=1}^{T} \Big(\gamma_t^{r+1}(j) \delta_{o_t k} - b_j(k) \gamma_t^{r+1}(j) \Big) \tag{II.16}$$

The objective function proposed by Singer and Warmuth (1996) minimizes the divergence between the old and new model parameters penalized by the negative log-likelihood of each subsequence multiplied by a fixed positive learning rate ($\eta > 0$):

$$\lambda^{r+1} = \arg\min_{\lambda} \Big(KL(\lambda^{r+1}, \lambda^r) - \eta \ell_T(\lambda^{r+1}) \Big) \tag{II.17}$$

The Kullback-Leibler (KL) divergence (or relative entropy) is defined between two probability distributions $P_{\lambda^{(k)}}(o_{1:t})$ and $P_\lambda(o_{1:t})$ by:

$$KL(P_{\lambda^r} \parallel P_\lambda) \;\; = \;\; \sum_t \Pr_{\lambda^r}(o_{1:t}) \log \frac{\Pr_{\lambda^r}(o_{1:t})}{\Pr_\lambda(o_{1:t})} \tag{II.18}$$

KL is always non-negative and attains its global minimum at zero for $\Pr_\lambda \to \Pr_{\lambda^r}$. This optimization is based on the exponentiated gradient framework, therefore the parameters

constraints are respected implicitly. After processing each subsequence of observations, the model parameters are updated using the conditional state densities and the derivatives of the log-likelihood ((II.10) and (II.11)):

$$a_{ij}^{r+1} = \frac{1}{Z_1} a_{ij}^r e^{\left(-\frac{\eta}{\sum_{t=1}^{T} \gamma_t^{r+1}(i)} \frac{\partial \ell_T(\lambda^{r+1})}{\partial a_{ij}} \right)} \tag{II.19}$$

$$b_j^{r+1}(k) = \frac{1}{Z_2} b_j^r(k) e^{\left(-\frac{\eta}{\sum_{t=1}^{T} \gamma_t^{r+1}(j)} \frac{\partial \ell_T(\lambda^{r+1})}{\partial b_j(k)} \right)} \tag{II.20}$$

where Z_1 and Z_2 are normalization factors.

The idea proposed by Ryden (1998) is to consider successive subsequences of T observations taken from a data stream, $\boldsymbol{o}_r = \{o_{(r-1)T+1}, \ldots, o_{rT}\}$, as independent of each other. This assumption reduces the extraction of information from all the previous observations to a data-segment of length T. In fact, this has been considered implicitly with all the above techniques that process multiple subsequences of T observations. To enforce parameters stochastic constraints (II.1), a projection ($\mathbb{P}_G$) on a simplex is suggested, which updates all but one of the parameters in each row of the matrices. At each iteration, the recursion is given (without matrix inversion)

$$\lambda^{r+1} = \mathbb{P}_G \left(\lambda^r + \eta_r h(\boldsymbol{o}_{r+1} \mid \lambda^r) \right) \tag{II.21}$$

where $h(\boldsymbol{o}_{r+1} \mid \lambda^r) = \nabla_{\lambda^r} \log \Pr(\boldsymbol{o}_{r+1} \mid \lambda^r)$ and $\eta_r = \eta_0 r^{-\rho}$ for some positive constant η_0 and $\rho \in (\frac{1}{2}, 1]$. It was shown to converge almost surely to the set of Kuhn–Tucker points for minimizing the Kullback-Leibler divergence $KL_k(\lambda^r \parallel \lambda^{(true)})$ defined in (II.18). KL attains its global minimum at $\lambda^r \to \lambda^{(true)}$, provided that the HMM is identifiable, therefore the subsequence must contains at least two symbols ($T \geq 2$).

On-line learning algorithms are therefore proposed for situations where a long (ideally infinite) sequences of observation are provided for training. In this paper however, these techniques are applied in an incremental fashion, where algorithms are allowed to con-

verge over several iterations of each new training block. However, when a new block is provided all learning rates are reset.

II.4 Experimental Methodology

The University of New Mexico (UNM) data sets are commonly used for benchmarking ADS based on system calls sequences. Normal data are collected from a monitored process in a secured environment, while testing data are the collection of the system calls when this process is under attack (Warrender et al., 1999). Since it is very difficult to isolate the manifestation of an attack at the system call level, the UNM test sets are not labeled. Therefore, in related work, intrusive sequences are usually labeled in comparison with the normal subsequences, (e.g., using STIDE). This labeling process leads to a biased evaluation of techniques, which depends on both training data size and detector window size.

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training, and labeled data) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations.

Inspired by the work of Tan and Maxion (Maxion and Tan, 2000; Tan and Maxion, 2003), the data generator is based on the Conditional Relative Entropy (CRE) of a source. It is defined as the conditional entropy divided by the maximum entropy (*MaxEnt*) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE can be computed by:

$$CRE = \frac{-\sum_x p(x) \sum_y p(y \mid x) \log p(y \mid x)}{MaxEnt} \tag{II.22}$$

where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a subsequence of system calls, the conditional probability, $p(y \mid x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $M = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly subsequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly subsequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly subsequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test[4].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE = 1$) is generated, using the same alphabet size Σ, and segmented into subsequences of a desired length using a sliding window with a fixed size of AS. Then, the original generative Markov chain is used to compute the likelihood of each subsequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol transitions in the subsequence of size AS. This ensures that the anomalous subsequences

[4]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occurs in high frequency over a short period of time.

of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into those subsequences at random according to a mixing ratio.

In the presented experiments, a normal data sequence of length $1,600$ symbols is produced using a Markov model with an irregularity index $CRE = 0.4$, and segmented using a sliding window of a fixed size, $DW = 8$ (Khreich et al., 2009b). The data are then divided into 10 blocks, D_i, for $i = 1, \ldots 10$, each comprises $R = 20$ subsequences. A test set of 400 subsequences each of size $AS = 8$ is prepared as described above. It comprises 75% of normal and 25% of anomalous data. Each block of the normal data D_i is divided into blocks of equal size – one is used for training (D_i^{train}) and the other for validation (D_i^{valid}), which is used to reduce the overfitting effects (hold-out validation).

For batch algorithms (BW-batch and GD-batch), successive blocks for training are accumulated and training is restarted each time a new block is presented. That is, the HMMs are first trained and validated on the first block of data (D_1^{train},D_1^{valid}). The training is then restarted, by re-initializing the models at random, and performing the batch algorithms using the accumulated blocks of data ($D_1^{train} \cup D_2^{train}$,$D_1^{valid} \cup D_2^{valid}$), and so on. In contrast, the incremental algorithms resume training by starting with the corresponding models produced from the previous block and by using the presented block only (D_i^{train}, D_i^{valid}). The stopping criterion for the batch and the incremental algorithms is set to a maximum of 100 iterations or to when the log-likelihood remains constant for at least 10 iterations for the validation data. On-line learning algorithms are not allowed to iterate on successive blocks of data. In this case, all learning rates are optimized and reset with the presentation of each new block. In all cases, the algorithms are applied to

an ergodic (fully connected) HMM with eight hidden states ($N = 8$), and are initialized with the same random model. For each algorithm, the model that produced the highest log-likelihood value on the validation data is selected for testing.

The log-likelihood of the test data (normal + anomalous) are then evaluated using the forward algorithm. By sorting the test subsequences decreasing by these log-likelihood values, and updating the true positive rate (tpr) and the false positive rate (fpr) while moving down the list, results in a Receiver Operating Characteristics (ROC) curves (Fawcett, 2006). The ROC curve depicts the trade-off between the tpr – the number of anomalous subsequences correctly detected over the total number of anomalous subsequences – and the fpr – the number of normal subsequences detected as anomalous over the total number of normal subsequences in the test set. The Area Under the ROC Curve (AUC) is used as a measure of performance. $AUC = 1$ means a perfect separation between normal and anomalous ($tpr = 100\%, fpr = 0\%$), while $AUC = 0.5$ means a random classification.

This procedure is replicated ten times with different training, validation and testing sets, and the resulting AUCs are averaged and presented along with their standard deviations (error bars). Although not show in this paper, various experiments have been conducted using different values of N, CRE, DW and Σ. These experiments produced similar results and hence the below discussion hold.

II.5 Results

The EM-based algorithm (Mizuno (Mizuno et al., 2000)), and the gradient-based ones (Baldi (Baldi and Chauvin, 1994), Singer (Singer and Warmuth, 1996), and Ryden (Ryden, 1998)) are first applied in an on-line learning approach as originally proposed by the authors. Figure AII.2 presents the average AUC achieved for on-line techniques for each block of training data. The results of the batch BW and GD algorithms are presented for reference.

The performances of the on-line algorithms tend to be equivalent with the increase of data. This is also confirmed by the statistical tests, conducted at the final block of data as shown in Figure AII.4. However, at the beginning, when few blocks of data

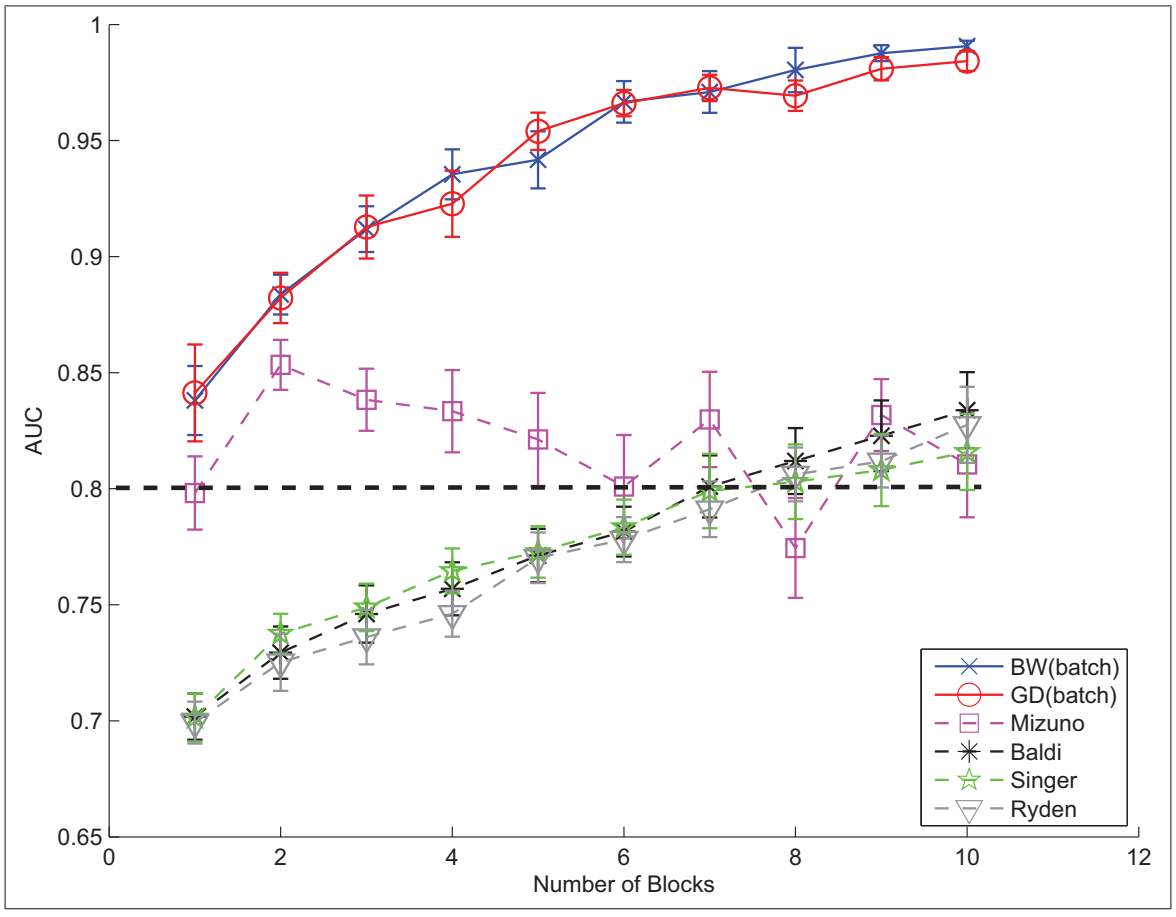


Figure AII.2 Average AUC of on-line learning techniques vs the amount of training data that are used to train an ergodic HMM with $N = 8$

are presented, the EM-based algorithm performs better than gradient-based ones. Due to one view of the data, on-line algorithms require a large amount of data to achieve good performance. Theoretically, an infinite amount of data is assumed when trying to prove the convergence of such on-line algorithms. Among the presented techniques, only Ryden provided convergence analysis and proof of consistency (Ryden, 1998). The average performances of the batch algorithms are equivalent and increase with the number of blocks. This is expected, since the batch algorithms are allowed to iterate on the accumulated data, they have therefore a global (backward) view.

Figure AII.3 presents the averaged AUC achieved by incremental techniques for each block of training data. The average AUC of the incremental algorithms are lower than the performance of batch and higher than that of the on-line ones. In fact, these algorithms have a local view of the presented data. Although they are allowed to learn the new data through several iterations, there is a loss of information form the previously learned data. This is usually controlled with the decaying learning rate, which assigns a weight to the past information with reference to the future data contribution.

Among the incremental algorithms, statistical testing shows that Baldi's algorithm (Baldi and Chauvin, 1994) achieved slightly superior performance than the others as shown in Figure AII.4. This is possibly related to the short length of training subsequences ($DW = 8$). The other incremental algorithms may require larger subsequences to accumulate enough information from each one before updating the model parameters. For instance, Ryden (1998) suggests using a minimum subsequence length of $DW = 20$.

However, the stochastic nature of the incremental algorithms allows them to escape local minima. This important characteristic can be shown when both batch and incremental algorithms are trained on the same data. For instance, this is illustrated when learning the first block in Figure AII.3. It can be seen that incremental algorithms are capable of producing superior results to their batch counterparts. Therefore, they may be used as an alternative for batch training, especially that they converge faster than the batch

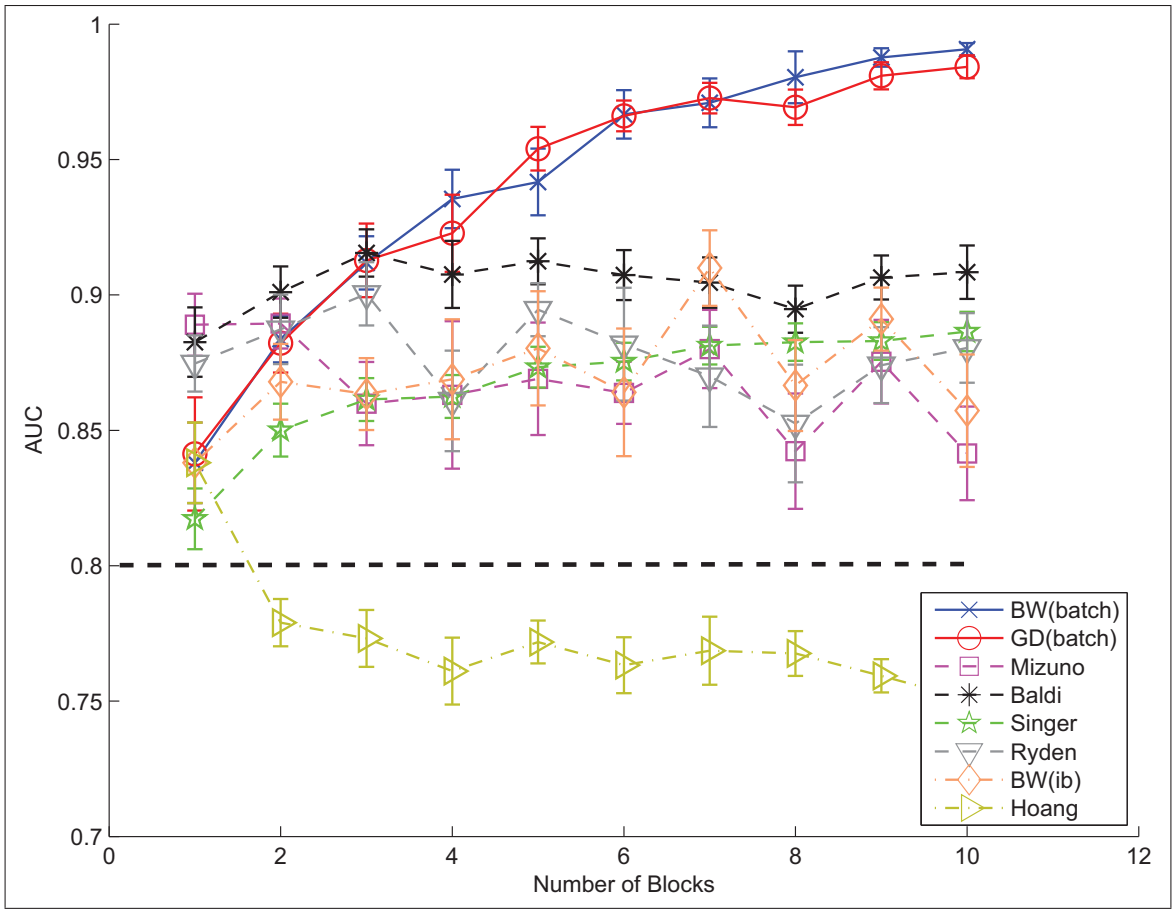


Figure AII.3 Average AUC of incremental learning techniques vs the amount of training data that are used to train an ergodic HMM with $N = 8$

algorithms since they update the model after each subsequence and hence exploit new information faster.

Figure AII.3 also presents the results of the BW incremental batch, BW(ib) for reference. That is, when learning D_1, BW is initialized with a random HMM and the algorithm is applied until it converges. For the subsequent block D_2, BW is then initialized with λ_1. The performance of BW(ib) indicates data corruption since it is prone to get stuck in a local minimum from the previous block. Interestingly, this straightforward EM-based solutions produced similar results to Mizuno (Mizuno et al., 2000). This indicates that the learning rates employed by the latter during the experiments may be better optimized to escape local minima.

In addition, Figure AII.3 includes another incremental approach based on learning an HMM for each new block of data then merging it with old ones using weight-averaging (Hoang and Hu, 2004). This learn and merge approach performed statistically worse than most of the incremental techniques as shown in Figure AII.4. This is due to averaging ergodic HMMs since the states order may be mixed up between the HMM trained on the first block and that trained on the second block of data.

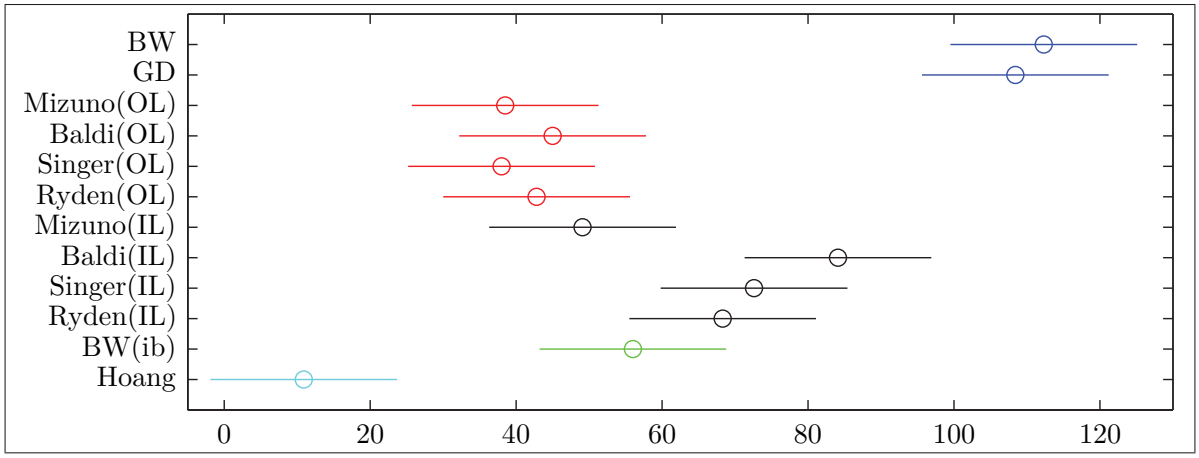


Figure AII.4 Kruskall-Wallis (one-way analysis of variance) statistical test for batch, on-line (OL) and incremental learning (IL) algorithms after processing the final block D_{10} of data

Table AII.1 compares the time and memory complexity of EM-based and gradient-based algorithms each processing a subsequence of T observations and then updating the model parameters. This represents the core operations required by all learning approaches. Time complexity represents the worst-case number of operations required for one iteration of EM-based and gradient-based algorithms. For both algorithms one iteration involves computing one forward and one backward pass with $o_{1:T}$. Memory complexity is the worst-case number of 32 bit words needed by the algorithms to store the required temporary variables in RAM. Since both algorithms rely on the FB or FFBS to compute the state conditional densities (Eqs. II.5 and II.6), therefore they require about the same computational time complexity, $\mathcal{O}(N^2T)$, and the same memory requirements,

$\mathcal{O}(NT)$. The additional computation time required by the gradient-based algorithms while updating HMM parameters stems from the re-parametrization (Section 3).

Table AII.1 Worst-case time and memory complexity analysis for EM- and gradient-based algorithms each processing a subsequence of T observations, with an N state HMM states and an alphabet of size $\Sigma = M$ symbols. This represents the core operations required by batch, on-line and incremental algorithms, the differences stem from iterating until convergence

Algo.	**Estimation**	**Time**			**Memory**
		# Multiplications	# Divisions	# Exponentiation	
EM-based	State Prob. (Eqs. II.5 & II.6)	$6N^2T+3NT-6N^2-N$	$N^2T+3NT-N^2-N$		$NT+N^2+2N$
	Transition Prob. (A)	N^2	N^2		N^2
	Emission Prob. (B)	NM	NM		NM
	Total	$6N^2T+3NT-5N^2$ $+NM-N$	$N^2T+3NT+MN-N$		$NT+2N^2+NM+2N$
Grad-based	State Prob. (Eqs. II.5 & II.6)	$6N^2T+3NT-6N^2-N$	$N^2T+3NT-N^2-N$		$NT+N^2+2N$
	Transition Prob. (A)	N^2	N^2	N^2	N^2
	Emission Prob. (B)	NM	MN	NM	NM
	Total	$6N^2T+3NT-5N^2$ $+NM-N$	$N^2T+3NT+MN-N$	N^2+NM	$NT+2N^2+NM+2N$

For a block of R subsequences, batch learning involves R times the computations of the state densities whereas only one update of the parameters is performed at each iteration. On the other hand, the on-line algorithms perform R times the computation of state densities and R times the update of the model, without iterating however. On-line algorithms are therefore the fastest in learning HMM parameters. At each iteration, the incremental algorithms require the same computational time need by the on-line ones. Accordingly, the incremental techniques require more computational time per iteration than the batch counterparts, however fewer iterations are required to converge. The memory complexity of the on-line algorithms is constant in time, $\mathcal{O}(NT)$, and also for the incremental ones though it is R times greater. However, for batch algorithms this value scales linearly with the number of the accumulated subsequences.

II.6 Conclusion

In this paper, the performance of several techniques is compared for on-line incremental learning of HMM parameters as new sequences of training data becomes available. These techniques are considered for updating host-based intrusion detection systems from sys-

tem calls to the OS kernel, but apply to other HMM-based detection systems that face the challenges associated with limited training data and environmental changes. Results have shown that techniques for incremental learning of HMM parameters can provide a higher level of discrimination than those for on-line learning, yet require significantly fewer resources than with batch training. Incremental learning techniques may provide a promising solution for adaptive intrusion detection systems. Future work includes in-depth investigation of the learning rates effects on performances, which are sensitive parameters especially for gradient-based techniques. Comparing these techniques with models combinations at the training level or responses fusion at the decision level is also an interesting direction to explore.

APPENDIX III

INCREMENTAL LEARNING STRATEGY FOR UPDATING HMM PARAMETERS

This Appendix presents an effective strategy to overcome the challenges faced when HMM parameters must be incrementally updated from a newly-acquired data.

Incremental learning refers to the ability of an algorithm to learn from new data that may become available after a classifier (or a model) has already been generated from a previously available data set. Incremental learning produces a sequence of hypotheses, for a sequence of training data sets, where the current hypothesis describes all data that have been seen thus far, but depends only on previous hypotheses and the current training data (Grossberg, 1988; Polikar et al., 2001). Incremental learning leads to a reduced storage requirement since there is no need for storing the data from previous training phases, after adapting HMM parameters. Furthermore, since training is performed only on the newly-acquired data, and not on all accumulated data, incremental learning would also lower the time and memory complexity required by the algorithm employed to train an HMM from new data.

However, incremental learning is typically faced with the stability-plasticity dilemma, which refers the problem of learning new information incrementally, yet overcoming the problem of catastrophic forgetting (Grossberg, 1988; Polikar et al., 2001). To sustain a high level of of generalization during future operations, HMM parameters should therefore be updated from new data without corrupting previously-acquired knowledge and without being subject to catastrophic forgetting. When new training data become available, standard batch techniques for estimating HMM parameters must restart the training procedure using all cumulative training data, as described in Chapter 2. Alternatively, the on-line learning techniques proposed in literature and surveyed in Chapter 2 assume infinite data sequence, and update HMM parameters continuously upon observing each

new sub-sequence or new symbol of observation. They typically employ some (fixed or decreasing) internal learning rate or decay function between the successive updates of HMM parameters, to help stabilizing the algorithm and adapt to the newly acquired information. In practice, both batch and on-line learning techniques lead to knowledge corruption when performed incrementally to update HMM parameters.

Figure AIII.1 illustrates the challenges caused by the stability-plasticity dilemma, when a batch or on-line learning algorithm is performed incrementally to update one HMM parameter. Assume that the training block D_2, which contains one or multiple sub-sequences, becomes available after an HMM(λ_1) has been previously trained on a block D_1 and deployed for operations. An incremental algorithm that optimizes, for instance, the log-likelihood function requires re-estimating the HMM parameters over several iterations until this function is maximized for D_2. The dotted curve in Figure AIII.1 represents the log-likelihood function of an HMM(λ_1) that has previously learned D_1, while the plain curve represents the log-likelihood function associated with HMM(λ_2) that has incrementally learned D_2 over the space of one model parameter (λ). Since λ_1 is the only information at hand from previous data D_1, the incremental training process starts from HMM(λ_1).

The optimization of HMM parameters depends on the log-likelihood function of HMM(λ_2) with respect to that of HMM(λ_1). If the log-likelihood function of HMM(λ_2) has some local maxima in the proximity of that of HMM(λ_1), the optimization of λ_2 may remain trapped in these local maxima and would not be able to accommodate the newly-acquired information from D_2. In this case, the model stability is privileged and its ability to learn new information becomes limited. For instance, if λ_1 was selected according to point (a) in Figure AIII.1, the optimization will probably lead to point (d), providing the same model parameters ($\lambda_2 \approx \lambda_1$). When HMM parameters remain trapped in the proximity of the same local maxima and old information is mostly conserved, the performance of HMM will decline over time. On the other hand, if the HMM parameters (λ_2) were able to escape the local maximum found by λ_1, λ_2 will be completely adapted to D_2, thereby

corrupting the previously-acquired knowledge, and compromising HMM performance. In this case, the plasticity of the model is privileged and its adaptability leads to a catastrophic forgetting. For instance, If λ_1 was selected as point (b), the optimization will lead to a better (g) or worse (e) solution on the log-likelihood function associated with D_2 alone.

The log-likelihood function depends on other factors than the starting point (λ_1), such as the size and regularity of newly-acquired data D_2 with respect to previously-learned data D_1, as well as on the learning algorithm. Given the same starting point λ_1, and assuming that a batch or on-line algorithm is employed to incrementally update HMM parameters from the same block of data D_2. In general, batch algorithms tend to privilege stability, while on-line algorithms tend to privilege plasticity. In fact, starting from λ_1, one iteration of a batch algorithm requires accumulating the sufficient statistic – the conditional state and joint state densities required to update HMM parameters, over

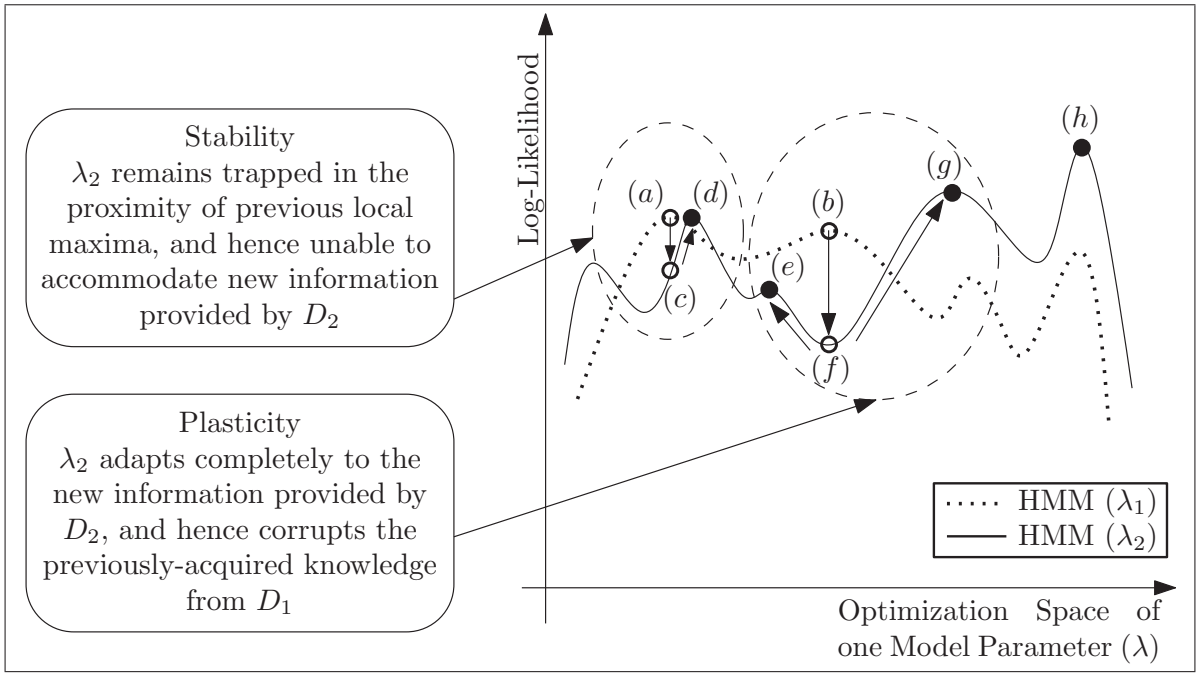


Figure AIII.1 An illustration of the degeneration that may occur with batch or on-line estimation of HMM parameters, when learning is performed incrementally on successive blocks data, each comprising unrepresentative amount of data providing incomplete view of phenomena

the entire sequence or sub-sequences in D_2, prior to updating λ_1. On the other hand, given the initial model parameters λ_1, a typical iteration of on-line algorithms involves a sequence of updates to λ_1 based on each observed symbol or each observed sub-sequence in D_2, until the last symbol or sub-sequence is reached. This added stochasticity induced by the early updates of HMM parameters according to on-line algorithms force the HMM (λ_1) to escape previous local maxima, and to achieve a faster and complete adaptation to D_2. However, the previously-acquired knowledge of the model may be compromised with the first few symbols or sub-sequences of D_2, leading thereby to a catastrophic forgetting.

In our previous work, described in Appendix II, some representative on-line learning techniques for HMM parameters are extended to incremental learning, by allowing them to iterate over each block of data and by resetting their internal learning rates when a new block is presented. The learning rates are employed to integrate the information from each new symbol or sub-sequences of observation with the previously-learned model. Although they have been shown to improve accuracy over time, optimizing the internal learning rates employed within these algorithms to trade off plasticity against stability is a challenging task. For one, information form all previously learned data contained within the current model should be balanced with new information contained in a symbol or sub-sequence of observation. Furthermore, model selection and stopping criteria become very difficult, because these on-line algorithms do not guarantee a monotonic improvement of the log-likelihood value at each update of parameters.

To alleviate these issues, the incremental learning strategy proposed in this appendix consists of employing a (global) learning rate at the iteration level. This aims at delaying the integration of newly-acquired information with the previously-learned model until at least on iteration over the new block of data is performed by the new model. In addition, the previously-learned model is kept in memory and weight-balanced with the new model over each iteration.

Assume that the HMM parameters (λ_{D_n}) obtained from the n^{th} data block D_n must be updated to accommodate the newly-acquired data from D_{n+1}. The learning process of $\lambda'_{D_{n+1}}$ starts from λ_{D_n}. At each iteration ($iter$), the parameters of the new HMM ($\lambda'_{D_{n+1}}$) are weight-balanced with those of previously-trained HMM (λ_{D_n}) and the new model parameters, which are iteratively updated from D_{n+1} ($\lambda^{(iter)}_{D_{n+1}}$), according to a learning rate η:

$$\lambda'_{D_{n+1}} \leftarrow \eta\lambda_{D_n} + (1-\eta)\lambda^{(iter)}_{D_{n+1}} \tag{III.1}$$

Several advantages are provided by retaining the previous model λ_{D_n} in memory during the current learning stage, and employing the learning rate η to reintegrate pre-existing knowledge from λ_{D_n} and the newly-acquired information from successive iterations of $\lambda^{(iter)}_{D_{n+1}}$ over D_{n+1}. For one, the previously-acquired knowledge contained in λ_{D_n} is no longer susceptible to be compromised by the insufficient information provided by the first symbol or sub-sequence of observation from D_{n+1}. The internal learning rates of on-line algorithms can be now optimize based on new data to stabilize the algorithm and ensure its converge smooth convergence, without worrying about corrupting λ_{D_n}. In addition, monitoring the improvement of log-likelihood values of the new data given the updated model parameters at each iteration becomes more tractable, which facilitates both stopping criteria of algorithms and selection of models for operations.

This incremental learning strategy is general in that it can be applied to any batch of on-line learning algorithm. The iterative procedure provided by Eq. III.1 does not specify the algorithm for local learning from the new block of data D_{n+1}. Therefore, updating $\lambda^{(iter)}_{D_{n+1}}$ is not restricted to on-line algorithms, other techniques such as batch learning algorithms for estimating HMM parameters can be also applied. Algorithm selection depends on the size and regularity of the data. When limited amount of data is provided for training within each new block, batch algorithms provide easier solution, as no internal learning rate is employed. However when abundant yet unrepresentative data is provided on-line algorithms may be employed due to their reduced storage requirements. However, an

effort is still required for optimizing the internal learning rates of these on-line algorithms to ensure convergence. In this case, the EFFBS algorithm proposed in Appendix I provides an alternative efficient solution, which requires no learning rate optimization.

In general, employing a fixed user-defined learning rate along with a validation strategy would maintaining the level of performance in static or slowly changing environments. As described in Section 2.5, various validation strategies may be employed, to reduce the overfitting effects and improve the generalization performance during operations. For instance, such as hold-out or cross-validation procedures, or when applicable accumulating and updating representative validation data set over time. However, this requires investigating some selection criteria for maintaining the most informative sequences of observations and discarding less relevant one. Other strategies for an automatic update of the learning rate based upon the magnitude of the performance change, during the incremental training and validation can be also applied (Kuncheva and Plumpton, 2008).

A version of this incremental learning strategy called incremental BW (IBW), has been applied in the proof-of-concept simulations conducted in Section 4.4. IBW employs the BW algorithm to update HMM parameters at each iteration, and uses a fixed learning rate η optimized during the first learning stage, with a 10-fold cross validation procedure for model selection.

APPENDIX IV

BOOLEAN FUNCTIONS AND ADDITIONAL RESULTS

IV.1 Boolean Functions

Figure AIV.1 presents all the distinct Boolean functions on two variables. Table (a) shows the four Boolean operations that depend on one variable (the inputs and their negations) along with the two constant operations (always positive and always negatives). Table (b) represents the ten operations that take on two variables. The first four are obtained from conjunction with some subset of its inputs negated. Similarly for the next four which are obtained from disjunction however. The last two (the XOR and its negation, EQV) are obtained with a "checkerboard" truth table. In general, for two inputs variables ($n = 2$) with two possible binary outputs, there are $2^{2^n} = 16$ distinct Boolean operations, whereas ten are effectively used for combining.

C_a	C_b	$\neg C_a$	$\neg C_b$	*One*	*Zero*
0	0	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	1	0

Depend on one variable — Constant operations

(a)

De Morgan's laws:

$$\neg(C_a \wedge C_b) = \neg C_a \vee \neg C_b$$
$$\neg(C_a \vee C_b) = \neg C_a \wedge \neg C_b$$

Symbols:

$\neg$	=	*NOT*
$\wedge$	=	*AND*
$\vee$	=	*OR*
$\oplus$	=	*XOR*
IMP	=	*Implication*
EQV	=	*Equivalence*

		AND			*NAND*	*OR*	*C_a IMP C_b*	*C_b IMP C_a*	*NOR*	*XOR*	*EQV*
C_a	C_b	$C_a \wedge C_b$	$\neg C_a \wedge C_b$	$C_a \wedge \neg C_b$	$\neg(C_a \wedge C_b)$	$C_a \vee C_b$	$\neg C_a \vee C_b$	$C_a \vee \neg C_b$	$\neg(C_a \vee C_b)$	$C_a \oplus C_b$	$\neg(C_a \oplus C_b)$
0	0	0	0	0	1	0	1	1	1	0	1
0	1	0	1	0	1	1	1	0	0	1	0
1	0	0	0	1	1	1	0	1	0	1	0
1	1	1	0	0	0	1	1	1	0	0	1

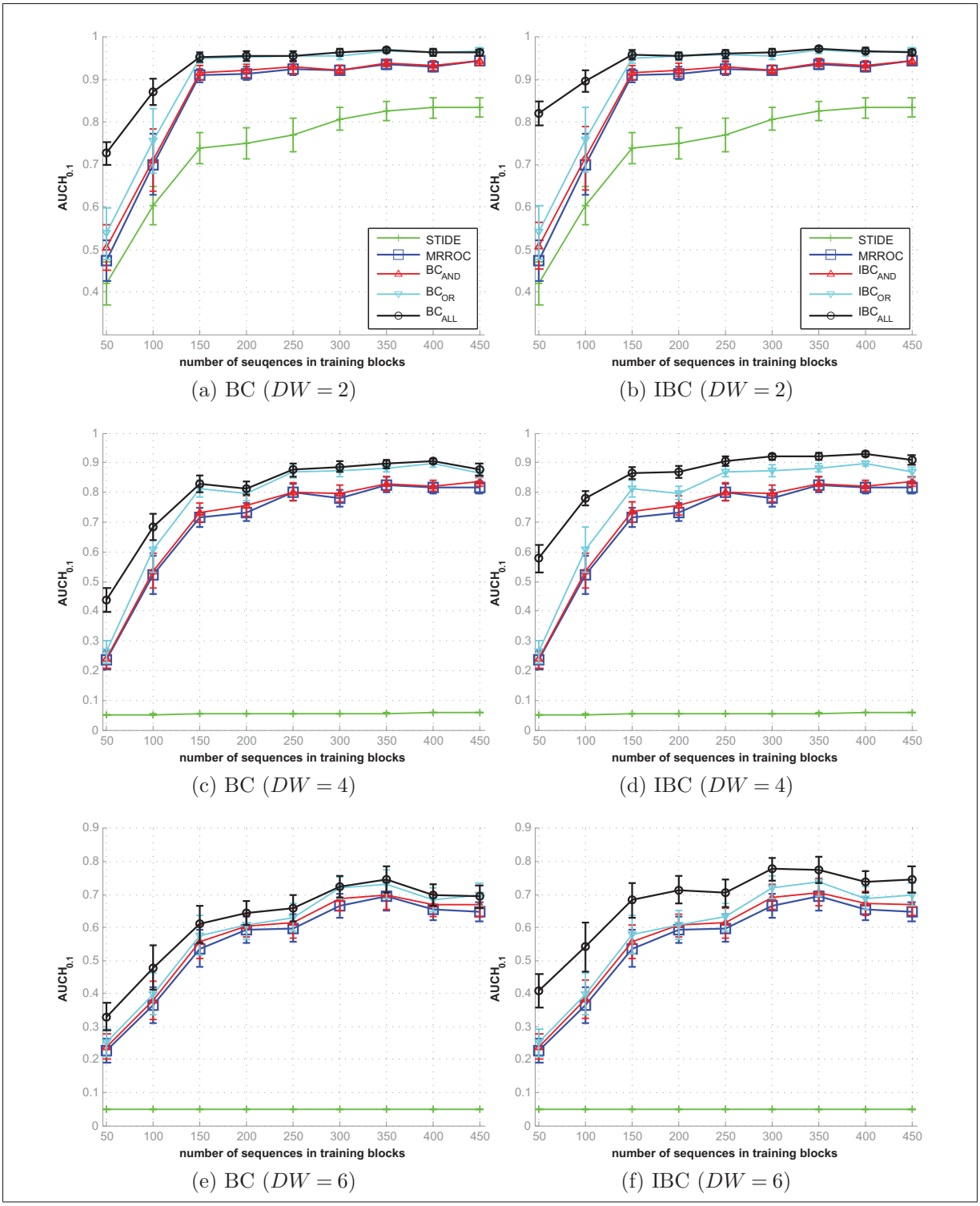
Called minterms: they represent the four miminum classes — Called maxterms: they represent the four maximum classes — Operations with a "checkerboard" truth table

(b)

Figure AIV.1 All distinct Boolean operations on two variables

IV.2 Additional Results

Figures AIV.2 and AIV.3 present the average performance in terms of the partial area under the convex hull for the range of $fpr = [0, 0.1]$ ($AUCH_{0.1}$), and the tpr at a fixed $fpr = 0.1$, respectively. These additional results are provided for a μ-HMM with three HMMs, each one trained with a different number of states ($N = 4, 8, 12$), and using the synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. These results complement those shown in Figure 3.8 (Section 3.6.2).

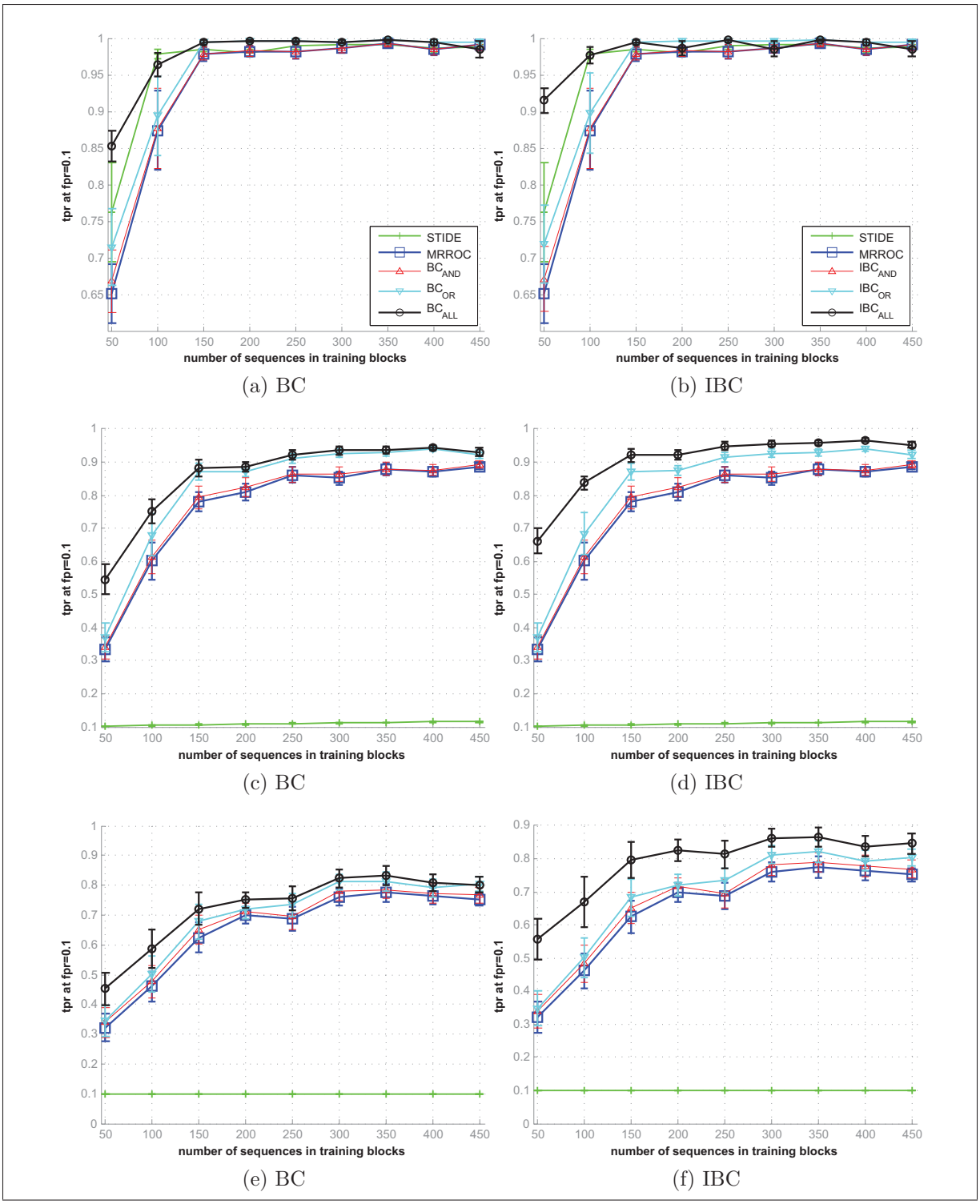


(a) BC ($DW = 2$) (b) IBC ($DW = 2$)

(c) BC ($DW = 4$) (d) IBC ($DW = 4$)

(e) BC ($DW = 6$) (f) IBC ($DW = 6$)

Figure AIV.2 Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. Average $AUCH_{0.1}$ values obtained on the test sets as a function of the number of training blocks (50 to 450 sequences) for a 3-HMM system, each trained with a different state ($N = 4, 8, 12$), and combined with the MRROC, BC and IBC techniques. Results are compared for various detector windows sizes ($DW = 2, 4, 6$). Error bars are standard deviations over ten replications



(a) BC (b) IBC

(c) BC (d) IBC

(e) BC (f) IBC

Figure AIV.3 Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. Average *tpr* values at $fpr = 0.1$ obtained on the test sets as a function of the number of training blocks (50 to 450 sequences) for a 3-HMM system, each trained with a different state ($N = 4, 8, 12$), and combined with the MRROC, *BC* and *IBC* techniques. Results are compared for various detector windows sizes ($DW = 2, 4, 6$). Error bars are standard deviations over ten replications

APPENDIX V

COMBINING HIDDEN MARKOV MODELS FOR IMPROVED ANOMALY DETECTION *

Abstract

In host-based intrusion detection systems (HIDS), anomaly detection involves monitoring for significant deviations from normal system behavior. Hidden Markov Models (HMMs) have been shown to provide a high level performance for detecting anomalies in sequences of system calls to the operating system kernel. Although the number of hidden states is a critical parameter for HMM performance, it is often chosen heuristically or empirically, by selecting the single value that provides the best performance on training data. However, this single best HMM does not typically provide a high level of performance over the entire detection space. This paper presents a multiple-HMMs approach, where each HMM is trained using a different number of hidden states, and where HMM responses are combined in the Receiver Operating Characteristics (ROC) space according to the Maximum Realizable ROC (MRROC) technique. The performance of this approach is compared favorably to that of a single best HMM and to a traditional sequence matching technique called STIDE, using different synthetic HIDS data sets. Results indicate that this approach provides a higher level of performance over a wide range of training set sizes with various alphabet sizes and irregularity indices, and different anomaly sizes, without a significant computational and storage overhead.

*This chapter is published in International Conference on Communications (ICC09)

V.1 Introduction

Intrusion Detection Systems (IDS) is used to identify, assess, and report unauthorized computer or network activities. Host-based IDSs (HIDS) are designed to monitor the host system activities and state, while network-based IDSs monitor network traffic for multiple hosts. In either case, IDSs have been designed to perform misuse detection (looking for events that match patterns corresponding to known attacks) and anomaly detection (detecting significant deviations from normal system behavior). In a HIDS for anomaly detection, operating system events are usually monitored. Since system calls are the gateway between user and kernel mode, most traditional host-based anomaly detection systems monitor deviation in system call sequences.

In general, anomaly detection systems seek to detect novel attacks, yet will typically generate false alarms mainly due to incomplete data for training, poor modeling, and difficulty in obtaining representative labeled data for validation. In practice, it is very difficult to acquire (collect and label) comprehensive data sets to design a HIDS for anomaly detection. Forrest et al. (1996) confirmed that short sequences of system calls are consistent with normal operation, and unusual burst will occur during an attack. Their anomaly detection system, called Sequence Time-Delay Embedding (STIDE), is based on segmenting and enumerating the training data into fixed-length *contiguous* sequences, using a fixed-size sliding window, shifted by one symbol. During operations, the sliding window scheme is used to scan the data for anomalies – sequences that are not found in the normal data.

Various anomaly detection techniques have been applied to learn the normal process behavior through system call sequences (Warrender et al., 1999). Among these, techniques based on discrete Hidden Markov Models (HMMs) (Rabiner, 1989) have been shown to produce slightly superior results, at the expense of training resource requirements. Estimating the parameters of an HMM requires the choice of the number of hidden states. In the literature on HMMs applied to anomaly detection (Gao et al., 2002; Hoang and

Hu, 2004; Hoang et al., 2003; Wang et al., 2004), the impact of the number of states on performance is often overlooked. It is typically chosen heuristically or empirically by selecting the number of states that provides the best performance on training data. Given incomplete training data, types of anomalies, and detector window (DW) sizes, a single "best" HMM does not provide a high level of performance over the entire detection space.

For a given data set, a multiple-HMMs (μ-HMMs) approach, where each model is trained using a different number of hidden states, allows to outperform any single best HMM. The responses of these HMMs are combined in the receiver operating characteristics (ROC) space, using the Maximum Realizable ROC (MRROC). This approach allows selecting operational HMMs independently from prior and class distributions, and cost contexts. Although the MRROC fusion has been applied to combine responses of different classifiers in fields such as medical diagnostics and image segmentation (Scott et al., 1998), it has never appeared in HIDS literature. In particular, it has never been proposed in HMM application to anomaly detection due to the overlooked impact on performance of the number of HMM states, and to its expensive training resource requirements, which are emphasized in this work. However, other fusion techniques such as weighted voting has been applied to combine HMMs trained on different features (Choy and Cho, 2000).

The objective of this paper consists in comparing the performance of μ-HMMs combined through the MRROC fusion, to that of a single best HMM and STIDE for detecting anomalies in system calls sequences. The impact on performance of using different training set sizes, anomaly sizes, and complexities of the monitored processes is assessed using ROC analysis (Fawcett, 2006). To this end, a synthetic simulator for normal data generation and anomaly injection has been constructed to overcome the issues encountered when experimenting with real data.

The rest of this paper is organized as follows. The next section describes the application of HMMs in anomaly-based HIDS. In Section V.3, the proposed μ-HMMs approach is

presented. Then, the experimental methodology in Section V.4 describes data generation, evaluation methods and performance metrics. Finally, simulation results are presented and discussed in Section V.5.

V.2 Anomaly Detection with HMM

A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms – a latent Markov chain having a finite number of states, and a set of observation probability distributions, each one associated with a state. At each discrete time instant, the process is assumed to be in a state, and an observation is generated by the probability distribution corresponding to the current state. HMMs are usually trained using the Baum-Welch algorithm (Baum et al., 1970) – a specialized expectation maximization technique to estimate the parameters of the model from the training data. Theoretical and empirical results have shown that, given an adequate number of states and a sufficiently rich set of observations, HMMs are capable of representing probability distributions corresponding to complex real-world phenomena in terms of simple and compact models. Given the trained model, the Forward algorithm (Baum et al., 1970) can be used to evaluate the likelihood of the test sequence. For further details regarding HMM the reader is referred to the extensive literature e.g., Rabiner (1989).

Estimating the parameters of a HMM requires the specification of the number of hidden states (N). Warrender et al. (1999) present the first application to anomaly-based HIDS with the University of New Mexico (UNM) data sets[1]. The authors trained an ergodic HMM on the normal data sequence for each process of the UNM data set. The number of states was selected heuristically. It is set roughly equal to the alphabet size – the number of unique system call symbols used by the process. Each HMM is then tested on the anomalous sequences, looking for unusual state transitions and/or symbol outputs according to a predefined threshold. Compared to other techniques, HMM produced slightly superior results in terms of average detection and false alarm rate, at the expenses

[1] `http://www.cs.unm.edu/~immsec/systemcalls.htm`

of expensive training resource requirements. Indeed, the time complexity of the Baum-Welch algorithm per iteration scales linearly with the sequence length and quadratically with the number of states. In addition, its memory complexity scales linearly with both sequence length and number of states.

Subsequent applications of HMMs to anomaly-based HIDS, e.g., (Gao et al., 2002; Hoang and Hu, 2004; Hoang et al., 2003; Wang et al., 2004), follow Warrender et al. (1999) in choosing the number of HMM states without further optimization. The authors in (Yeung and Ding, 2003) are among few who tried to investigate the effect of the number of states and the detector window size on the performance using UNM data sets. However only *unique* contiguous sequences are used for HMM training, which might affect the model parameters. Furthermore, as explained in Section V.4, the data sets are labeled using the STIDE matching technique which makes it as the ground truth for other classifiers.

V.3 Selection and Fusion of HMMs in the ROC Space

The ROC curve displays a trade-off between true detection (or true positive) rate and false alarm (or false positive) rate for all thresholds. In our context, the true detection rate is the number of anomalous sequences correctly detected over the total number of anomalous sequences, and the false alarm rate is the number of normal sequences detected as anomalous over the total number of normal sequences in the test set.

ROC analysis provides a useful tool for combining classifiers given a set of one- or two-class classifiers using the MRROC (Scott et al., 1998). The MRROC is a fusion technique that uses randomized decision rules and provides an overall classification system whose curve is the convex hull over all existing detection and false alarm rates. In contrast with traditional ROCs where only operational thresholds change when moving along the curve, the MRROC points may also change other model parameters or even switch models. All operating points on the MRROC are achievable in practice, using an exhaustive search, and are theoretically proved by application of the realizable classifier theory (Scott et al.,

1998). The procedure is to create from two existing classifiers C_a and C_b a composite one C_c whose performance in terms of its ROC, lies on the line segment connecting C_a and C_b. This is done by interpolating the responses of the existing classifiers as follows. Let (tpr_a, fpr_a) and (tpr_b, fpr_b) be the true positive and false positive rates of the existing classifiers C_a and C_b, and (tpr_c, fpr_c) those of the desired composite classifier C_c. The output of C_c for any input $O = \{o_1, \ldots, o_T\}$ from the test set, is a random variable that selects the output of C_a or C_b with probability:

$$\begin{aligned} \Pr(C_c = C_b) &= \frac{fpr_c - fpr_a}{fpr_b - fpr_a} \\ \Pr(C_c = C_a) &= 1 - \Pr(C_c = C_b) \end{aligned} \tag{V.1}$$

Instead of training the HMM with an arbitrary chosen number of states, our approach consists in training several HMMs over a range of $n = [N_{min}, N_{max}]$ values and combine their decisions in the ROC space using the MRROC. HMMs trained with various number of states capture different temporal structures of the data. As illustrated in Figure AV.1, for a desired operational system C_c the fusion is done by switching at random between the responses of neighboring models C_a and C_b on the convex hull curve according to eq. (V.1). This combination leads to the realization of the virtual desired model C_c, which is able to achieve in the least a higher performance than any existing models. Since only the HMMs touching the MRROC are potentially optimal, no others need be retained. In addition, this approach allows visualizing systems' performance and selecting operational HMMs independently from both prior and class distributions as well as cost contexts (Fawcett, 2006). As conditions change, the MRROC does not change; only the portion of interest will. This change is accounted for by shifting the operational system to another point on the facets of the MRROC. In contrast with the commonly used anomaly index measure in related work, this approach also permits to use the Area Under the ROC Curve (AUC), which has been proved an effective evaluation measure in contexts with variable misclassification costs or skewed data (Fürnkranz and Flach, 2003).

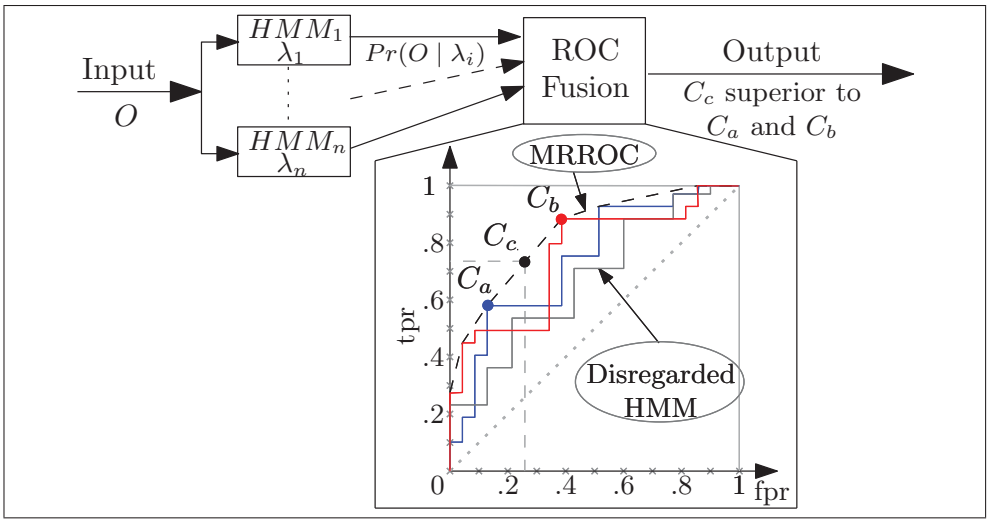


Figure AV.1 Illustration of the μ-HMMs using the MRROC fusion. For a particular operational system, the combined model may achieve a higher performance than any existing one. Models below the MRROC are disregarded

The time and memory complexity required for Baum-Welch training of n different HMMs for the μ-HMMs is comparable to that of training an HMM over a range of n different N values, and selecting the single best HMM (i.e., $\mathcal{O}(n.N.DW^2)$). During operations, the worst-case time complexity of the Forward-Backward algorithm to evaluate a sequence of data (for a given detection and false alarm rate) with the μ-HMMs solution is two times of the single best HMM solution (i.e., $\mathcal{O}(N.DW^2)$). For a specific detection and false alarm rate, the MRROC requires in the worst-case the responses of two HMMs to interpolate between two convex hull facets in the ROC space. The worst-case memory complexity required to store HMM parameters may be significantly higher than that of a single best HMM. However, HMMs with a ROC curve that does not touch the MRROC are suboptimal and may be discarded.

V.4 Experimental Methodology

The UNM data sets are commonly used for benchmarking anomaly detections based on system calls sequences. Normal data are collected from the monitored process in a secured environment, while testing data are the collection of the system calls when this process is under attack (Warrender et al., 1999). Since it is very difficult to isolate the manifestation of an attack at the system call level, the UNM test sets are not labeled. Therefore, in related work, intrusive sequences are usually labeled by comparing normal sequences, using STIDE matching technique. This labeling process considers STIDE response the ground truth, and leads to a biased evaluation and comparison of techniques, which depends on both training data size and detector window size.

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training, and labeled data) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations.

Inspired by the work of Tan and Maxion (Maxion and Tan, 2000; Tan and Maxion, 2002, 2003), the data generator is based on the Conditional Relative Entropy (CRE) of a source. It is defined as the conditional entropy divided by the maximum entropy (*MaxEnt*) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE can be computed by:

$$CRE = \frac{-\sum_x p(x) \sum_y p(y \mid x) \log p(y \mid x)}{MaxEnt}$$

where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a sequence of system calls, the conditional probability, $p(y \mid x)$, represents

the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $M = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly sequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly sequence that contains n-grams not present in the process normal data, or a *rare n-gram* anomaly sequence that contains n-grams that are infrequent in the process normal data and occurs in burst during the test[2].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE = 1$) is generated, using the same alphabet size Σ, and segmented into sub-sequences of a desired length using a sliding window with a fixed size of AS. Then, the original generative Markov chain is used to compute the likelihood of each sub-sequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol transitions in the sequence of size AS. This ensures that the anomalous sequences of size AS are not associated with the process normal behavior, and hence foreign n-gram anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare n-gram anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at

[2]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occurs in high frequency over a short period of time.

a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into this sequence at random according to a mixing ratio.

The experiments conducted in this paper cover a wide range of the parameters space: $\Sigma = 8 - 50$ symbols, training set size $= 100 - 1000,000$ symbols and $CRE = 0.3 - 0.8$, to approach as much as possible to real-world processes (Lee and Xiang, 2001). The sizes of injected anomalies are assumed equal to the detector window sizes $AS = DW = \{2, 4, 6, 8\}$, and different normal/anomalous ratios are considered for the test phase.

For each training set of size DW, different discrete-time ergodic HMMs are trained with various number of hidden states N. The number of symbols is taken equal to the process alphabet size. The iterative Baum-Welch algorithm is used to estimate HMM parameters (Baum et al., 1970). To reduce overfitting effects, the evaluation of the log-likelihood on an independent validation set is used as a stopping criterion. The training process is repeated ten times using a different random initialization to avoid local minima. Finally, the model that gives the highest log-likelihood value on validation data is selected. This procedure is replicated ten times with different training, validation and testing sets, and the results are averaged and presented along with the standard deviations.

After training an HMM, the log-likelihood of a test sub-sequences is evaluated using the forward algorithm (Rabiner, 1989). By sweeping all the decision thresholds, HMM evaluation results in an ROC curve for each value of N. In contrast, STIDE testing consists of comparing the test sub-sequences with its normal database, which gives a point in the ROC space. It should be noted that since in this paper, the detector window size is always considered equal to the anomaly size ($DW = AS$), STIDE detection rate is always 100% and only the false alarm rate varies. This is due to its blind regions – STIDE only misses anomalous sequences that are larger that the detector window size ($DW < AS$) and have all of its sub-sequences in STIDE normal database. The window

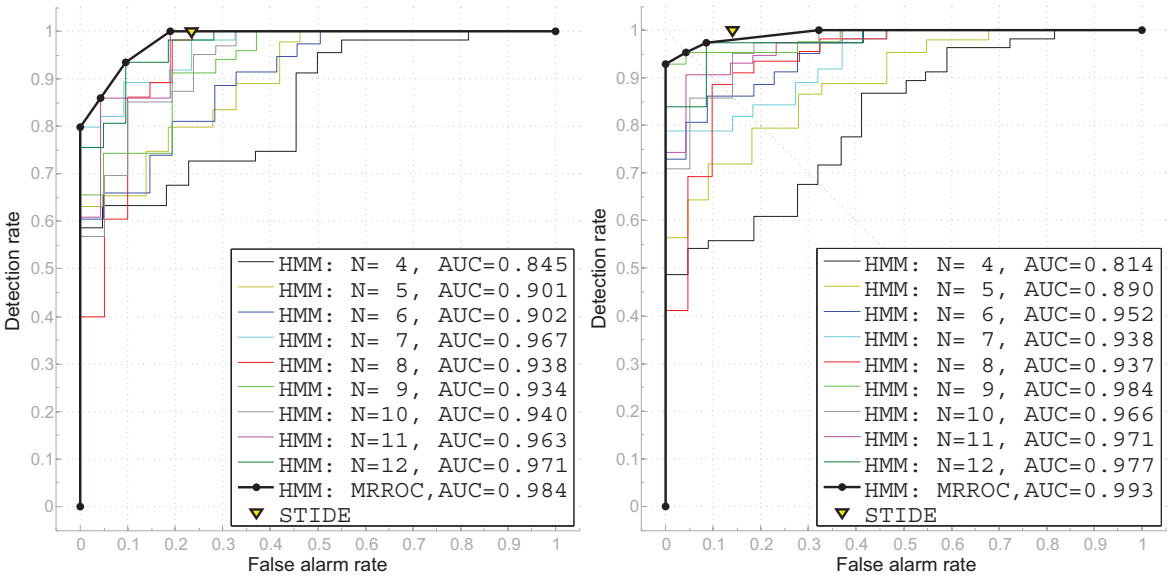
will slide on its sub-sequences that are all normal, without being able to discover that the whole sequence is anomalous (Maxion and Tan, 2000; Tan and Maxion, 2002, 2003).

V.5 Simulation Results

Due to space limitations only a limited number of results are presented. First, the results of a simpler scenario are illustrated in Figure AV.2 and AV.3, and then those of a more complex scenario are shown in Figure AV.4. The simpler scenario involves a relatively simple case with a small alphabet size $\Sigma = 8$ and a low irregularity index $CRE = 0.3$, while for the more complex scenario $\Sigma = 50$ symbols and the $CRE = 0.4$. For both scenarios, the presented results are for test sets that comprise 75% of normal and 25% of anomalous data. However, these results are consistent throughout the range of experiments not shown in this paper.

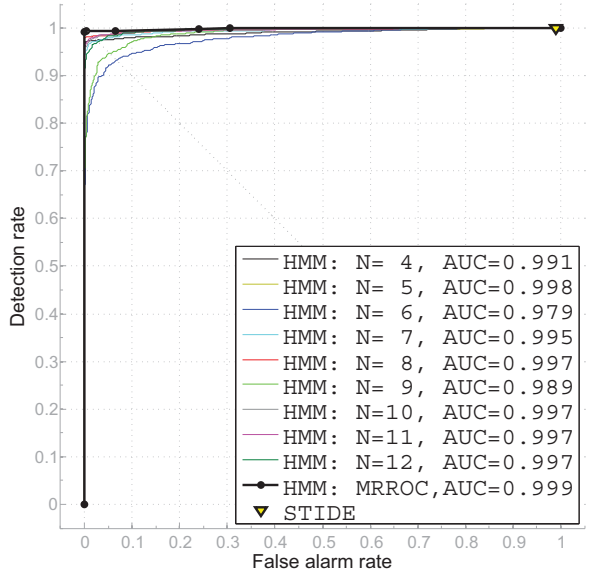
The ROC curves in Figure AV.2 show the impact of using different training set and anomaly sizes on the performance of STIDE and HMM, where HMMs are trained with different numbers of states N. Results in this figure indicate that the AUC of HMM grow with the training set size. As illustrated in Figure AV.2c, when the anomaly size increases, the search space grows exponentially (according to Σ^{AS}) which also expands both anomalous and normal partitions of the space. For instance, for $\Sigma = 8$ and $AS = 2$ the space of combinations comprises $8^2 = 64$ sequences, part of which is normal and the rest anomalous (depending on the CRE the data). By changing the anomaly size to $AS = 8$ the combinations grows to $8^8 = 16,777,216$ sequences, which imposes a much larger amount of training data for STIDE to memorize the normal space. The storage capacity and detection time associated with STIDE increases considerably. Accordingly, for a fixed training set size, the performance of STIDE is significantly degraded with the increase of the anomaly size. Nevertheless, any normal sequence that was not collected during training will be classified as anomaly, triggering false alarms in operational mode. This problem is usually mitigated by introducing an anomaly counter or index in local

regions according to an arbitrary threshold, which may however be exploited by an adversary.



(a) Training Set Size = 100 symbols, $DW = AS = 2$ symbols

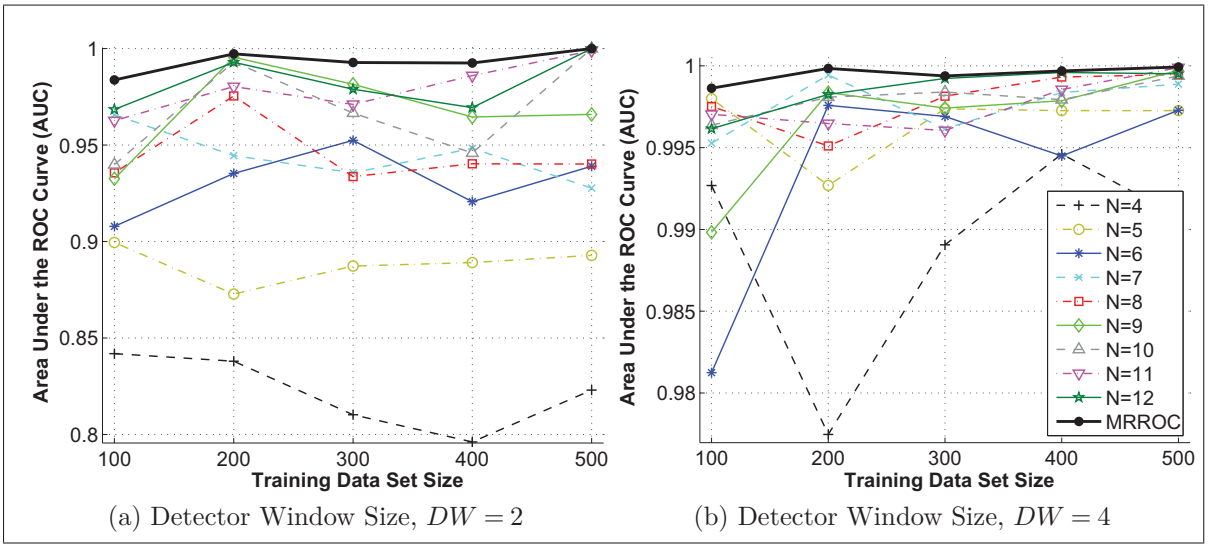
(b) Training Set Size = 300 symbols, $DW = AS = 2$ symbols



(c) Training Set Size = 100 symbols, $DW = AS = 8$ symbols

Figure AV.2 Illustration of the effect of training set sizes and anomaly sizes on the performance of STIDE, HMM with and the μ-HMMs using MRROC for the simple scenario. The HMMs were trained for a numbers of states $N \in [4, 12]$

In contrast, even when an HMM is trained on a relatively small data set, its discrimination capabilities increases with the anomaly size. This is due to the HMM abilities to capture dependencies in temporal patterns and to generalize in the case of unknown test sequences. Indeed, when evaluating the likelihood $Pr(O \mid HMM)$, an HMM computes the product of probabilities (over states transitions and emissions) for symbols in the sequence presented for testing. Anomalous sequences should lead to significantly lower likelihood values than normal ones with a well trained HMM. Therefore, with the increase of the anomaly size, the likelihood of anomalous sequences becomes smaller at a faster rate than normal ones, and hence increases HMM detection rate.

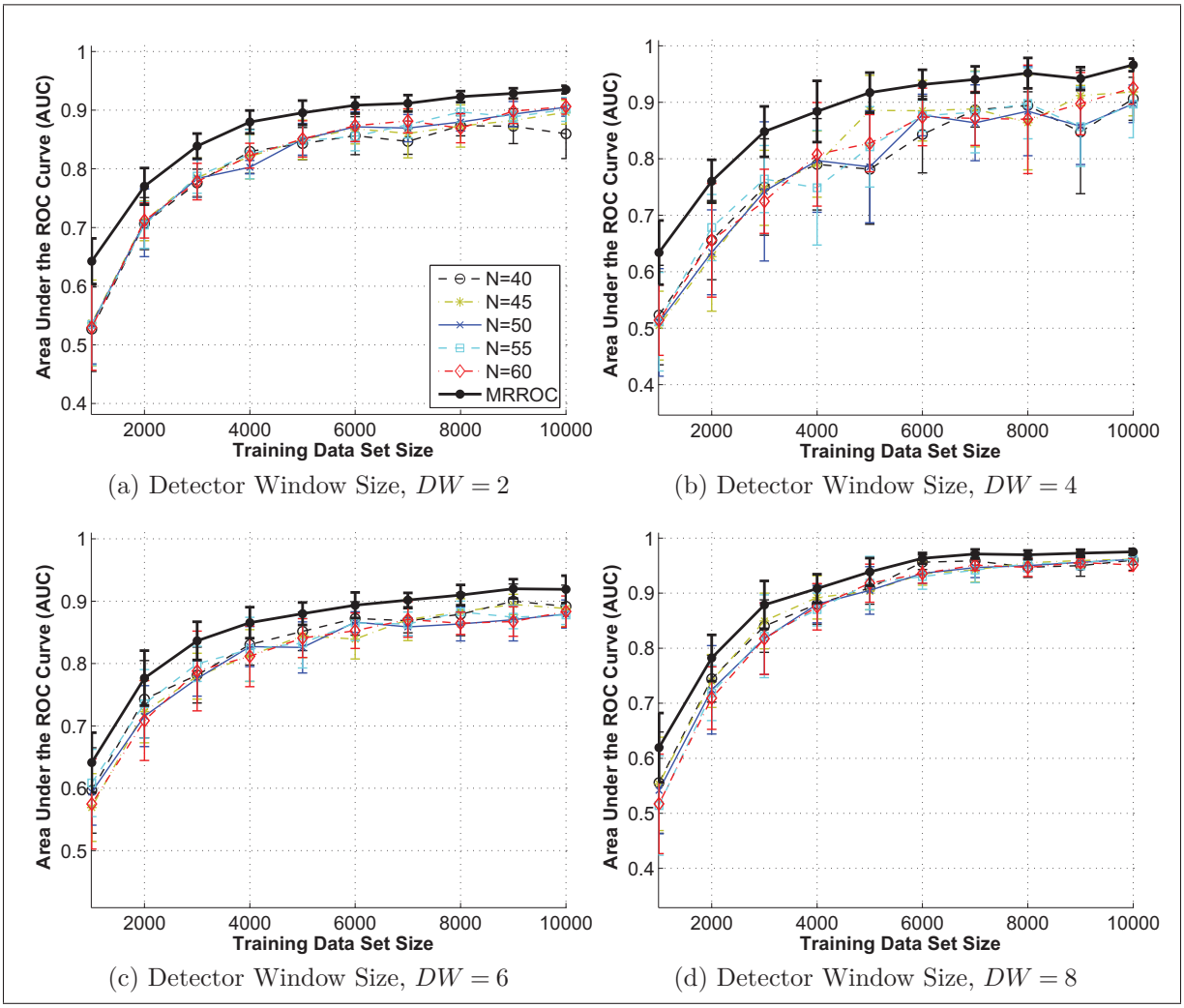


(a) Detector Window Size, $DW = 2$ (b) Detector Window Size, $DW = 4$

Figure AV.3 μ-HMMs vs HMM performance evaluation using the AUC for various training set sizes, DW and N ($\Sigma = 8$ and $CRE = 0.3$)

The impact of the number of HMM states N on performance is rarely addressed in the HIDS applications. Figures AV.2 and AV.3 illustrates this effect on AUC measures versus different amounts of training data and different detector window sizes DW. It can be seen that the value of N that provides the best performance depends on the training data set size and DW. More importantly, the common practice (in most application to HMMs for anomaly-based HIDS) of selecting the number of states equal to the alphabet size (e.g., $N = \Sigma = 8$ in Figure AV.3), does not provide a high level of performance over

the wide range of training set sizes and DW values. This effect is further illustrated in Figure AV.4 with the average AUC values (over ten independent replications) for the more complex scenario.



(a) Detector Window Size, $DW = 2$

(b) Detector Window Size, $DW = 4$

(c) Detector Window Size, $DW = 6$

(d) Detector Window Size, $DW = 8$

Figure AV.4 Average AUC of single HMMs and the μ-HMMs with MRROC combination for various training set sizes, DW and N values ($\Sigma = 50$ and $CRE = 0.4$). Error bars are standard deviations

The optimal number of states does not exist over the whole range of detection space to design a "single best" HMM. Combining the responses of μ-HMMs with the MRROC fusion is shown to achieve the highest over all performance over anyone of the HMMs alone. Furthermore, the resulting convex hull (from the MRROC combination) provides a smoother curve than any existing single model. The operational system may therefore

be changed according to the desired detection and false alarm rate without compromising the performance. In fact, moving along the convex hull curve allows switching to another model or interpolating the responses of two models. In contrast, as illustrated in Figure AV.2a and b, using a single HMM typically results a staircase-shaped ROC curve and this effect is even worse in real-world cases since the anomalies are relatively rare with reference to normal data.

Finally, in contrast with STIDE, HMM and the μ-HMMs are capable of detecting different anomaly sizes with the same detector window size. This impact along with the combination of HMMs trained with different window sizes is being characterized and will appear in future work.

V.6 Conclusion

This paper presents a multiple-HMMs approach, where each HMM is trained using a different number of hidden states capturing different temporal structures of the data. The HMM responses are then combined in the ROC space according to the MRROC technique. Results have shown that the overall performance of the proposed μ-HMMs approach increased considerably over a single best HMM and STIDE. In addition, this combination provides a smooth convex hull for composite operational systems without significantly increasing the computational overhead. Future work also involves characterizing the performance of the μ-HMMs approach using for instance the UNM data and observing the impact of combining HMMs trained with different window sizes to detect various anomaly sizes.

BIBLIOGRAPHY

Abouzakhar N. and Manson G.A., 2004. Evaluation of intelligent intrusion detection models. *International Journal of Digital Evidence*, volume 3.

Alanazi Hamdan O., Noor Rafidah Md, Zaidan B. B., and Zaidan A. A., February 2010. Intrusion detection system: Overview. *Journal of Computing*, 2(2):130–133.

AlephOne, November 1996. Smashing the stack for fun and profit. Online. *Phrack Magazine*, 7(14). URL `http://www.phrack.com/issues.html?issue=49&id=14`. Accessed December 10, 2010.

Alessandri Dominique, Cachin Christian, Dacier Marc, Deak Oliver, Julisch Klaus, Randell Brian, Riordan James, Tscharner Andreas, Wespi Andreas, and Wüest Candid, September 2001. Towards a taxonomy of intrusion detection systems and attacks. Malicious- and Accidental-Fault Tolerance for Internet Applications, MAFTIA Deliverable D3 EU Project IST-1999-11583, IBM, Zurich, Switzerland.

Ali Kamal M. and Pazzani Michael J., 1996. Error reduction through learning multiple descriptions. *Machine Learning*, 24:173–202.

Anderson B.D.O., 1999. From Wiener to hidden Markov models. *IEEE Control Systems Magazine*, 19(3):41–51. ISSN 0272-1708.

Anderson Debra, Frivold Thane, and Valdes Alfonso, May 1995. Next-generation intrusion detection expert system (nides): A summary. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International.

Anderson James P., April 1980. Computer security threat monitoring and surveillance. Technical Report 79F26400, James P. Anderson Co., Fort Washington.

Arapostathis Aristotle and Marcus Steven I., 1990. Analysis of an identication algorithm arising in the adaptive estimation of Markov chains. *Mathematics of Control, Signals and Systems*, 3(1):1–29.

Askar M. and Derin H., 1981. A recursive algorithm for the bayes solution of the smoothing problem. *IEEE Transactions on Automatic Control*, 26(2):558–561.

Axelsson Stefan, March 2000. Intrusion detection systems: A survey and taxonomy. Technical Report 99–15, Chalmers University.

Bahl Lalit R., Jelinek Frederick, and Mercer Robert L., 1982. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-5*, 2:179–190.

Baldi Pierre and Chauvin Yves, 1994. Smooth on-line learning algorithms for hidden Markov models. *Neural Computation*, 6(2):307–318. ISSN 0899-7667.

Banfield Robert, Hall Lawrence, Bowyer Kevin, and Kegelmeyer W., 2003. A new ensemble diversity measure applied to thinning ensembles. *Multiple Classifier Systems*, 2709:306–316.

Barreno Marco, Cardenas Alvaro, and Tygar Doug, January 2008. Optimal ROC for a combination of classifiers. *Advances in Neural Information Processing Systems (NIPS), 20*.

Baum Leonard E., 1972. An inequality and associated maximization technique in statistical estimation for probalistic functions of Markov processes. *Inequalities*, III: New York: Academic. (Proc. 3rd Symp., Univ. Calif., Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin).

Baum Leonard E. and Petrie Ted, 1966. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37:1554–1563.

Baum Leonard E., Petrie G. Soules, and Weiss N., 1970. A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Beal Matthew J., Ghahramani Zoubin, and Rasmussen Carl E., 2002. The infinite Hidden Markov Model. *Advances in Neural Information Processing Systems (NIPS), 14*.

Bengio Y., 1999. Markovian models for sequential data. *Neural Computing Surveys*, 2: 129–162. ISSN 1093-7609.

Benveniste Albert, Priouret Pierre, and Métivier Michel, 1990. *Adaptive algorithms and stochastic approximations*. Springer-Verlag New York, Inc., New York, NY, USA. ISBN 0-387-52894-6.

Bickel Peter J., Ritov Yaacov, and Ryden Tobias, 1998. Asymptotic normality of the maximum-likelihood estimator for general hidden Markov models. *Annal of Statistics*, 26(4):1614–1635.

Bilmes Jeff A., 2002. What HMMs can do. Technical Report UWEETR-2002-0003, Dept of EE, University of Washington Seattle WA.

Black Michael A. and Craig Bruce A., 2002. Estimating disease prevalence in the absence of a gold standard. *Statistics in Medicine*, 21(18):2653–2669.

Blunden Bill, 2009. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Wordware Publishing.

Bottou Léon, 2004. Stochastic learning. Bousquet Olivier and von Luxburg Ulrike, editors, *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin.

Bradley Andrew P., 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.

Brand M. and Kettnaker V., 2000. Discovery and segmentation of activities in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):844–851.

Bray Rory, Cid Daniel, and Hay Andrew, March 2008. *OSSEC Host-Based Intrusion Detection Guide*. SYNGRESS.

Breiman Leo, August 1996. Bagging predictors. *Machine Learning*, 24(2):123–140.

Brown G., Wyatt J., Harris R., and Yao X., March 2005. Diversity creation methods: A survey and categorisation. *Journal of Information Fusion*, 6(1):5–20.

Brugger S. T., Kelley M., Sumikawa K., and Wakumoto S., April 2001. Data mining for security information: A survey. Technical Report UCRL-JC-143464, U.S. Department of Energy, Lawrence Livermore National Laboratory., Philadelphia, PA.

Bulla Jan and Berzel Andreas, January 2008. Computational issues in parameter estimation for stationary hidden Markov models. *Computational Statistics*, 23(1): 1–18.

Cappe O., 2009. Online EM algorithm for hidden Markov models. *preprint*.

Cappe O. and Moulines E., 2005. Recursive computation of the score and observed information matrix in hidden Markov models. *IEEE/SP 13th Workshop on Statistical Signal Processing*, pages 703–708.

Cappe O., Buchoux V., and Moulines E., 1998. Quasi-Newton method for maximum likelihood estimation of hidden Markov models. *Proceedings of the 1998 IEEE International Conference onAcoustics, Speech, and Signal Processing, ICASSP98*, volume 4, pages 2265–2268.

Cappe Olivier, March 12 2001. Ten years of HMMs. Available online: http://www.tsi.enst.fr/ cappe/docs/hmmbib.html. Accessed December 10, 2010.

Cappe Olivier, Moulines Eric, and Ryden Tobias, 2005. *Inference in Hidden Markov Models*. Series in Statistics. Springer-Verlag, New-York.

Caragea D., Silvescu A.., and Honavar V., 2001. Towards a theoretical framework for analysis and synthesis of agents that learn from distributed dynamic data sources. *Emerging Neural Architectures Based on Neuroscience*, pages 547–559. Springer-Verlag.

Cardenas Alvaro A., Ramezani Vahid, and Baras John S., 2003. HMM sequential hypothesis tests for intrusion detection in MANETs. Technical Report ISR TR 2003-47, Department of Electrical and Computer Engineering and Institute for Systems Research University of Maryland College Park, MD, 20740 USA.

Caruana Rich, Niculescu-Mizil Alexandru, Crew Geoff, and Ksikes Alex, 2004. Ensemble selection from libraries of models. *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 18, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-5.

Chandola Varun, Banerjee Arindam, and Kumar Vipin, 2009a. Anomaly detection for discrete sequences: A survey. Technical Report TR 09-015, University of Minnesota, Department of Computer Science and Engineering.

Chandola Varun, Banerjee Arindam, and Kumar Vipin, July 2009b. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58. ISSN 0360-0300.

Chang R. and Hancock J., 1966. On receiver structures for channels having memory. *IEEE Transactions on Information Theory*, 12(4):463–468. ISSN 0018-9448.

Chen Hao, Wagner David, and Dean Drew, 2002. Setuid demystified. *USENIX Security Symposium*, pages 171–190.

Chen Yu-Shu and Chen Yi-Ming, 2009. Combining incremental hidden Markov model and Adaboost algorithm for anomaly intrusion detection. *CSI-KDD '09: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, pages 3–9, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-669-4.

Cho Sung-Bae and Han Sang-Jun, 2003. Two sophisticated techniques to improve hmm-based intrusion detection systems. Vigna Giovanni, Kruegel Christopher, and Jonsson Erland, editors, *Recent Advances in Intrusion Detection*, volume 2820 of *Lecture Notes in Computer Science*, pages 207–219. Springer Berlin / Heidelberg.

Choy Jongho and Cho Sung-Bae, 2000. Intrusion detection by combining multiple HMMs. *PRICAI 2000 Topics in Artificial Intelligence*, 1886:829–829.

Churbanov Alexander and Winters-Hilt Stephen, 2008. Implementing EM and viterbi algorithms for hidden Markov model in linear memory. *BMC Bioinformatics*, 9 (224):–.

Cohen William W., July 1995. Fast effective rule induction. Prieditis Armand and Russell Stuart, editors, *Proceeding of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 1995. Morgan Kaufmann. ISBN 1-55860-377-8.

Collings Iain B., Krishnamurthy Vikram., and Moore Jhon B., December 1994. On-line identication of hidden Markov models via recursive prediction error techniques. *IEEE Transactions on Signal Processing*, 42(12):3535–3539.

Collings I.B. and Ryden T., 1998. A new maximum likelihood gradient algorithm for on-line hidden Markov model identification. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP98*, volume 4, pages 2261–2264.

Coppersmith D. and Winograd S., 1990. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9:251–280.

Daugman John, 2000. Biometric decision landscapes. Technical Report UCAM-CL-TR-482, Universtity of Cambridge, UK.

De Boer Pieter and Pels Martin, 2005. Host-based intrusion detection systems. Technical Report 1.10, Faculty of Science, Informatics Institute, University of Amsterdam.

Debar H., Becker M., and Siboni D., May 1992. A neural network component for an intrusion detection system. *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 240–250.

Debar Herve, Dacier Marc, and Wespi Andreas, 2000. Revised taxonomy for intrusion-detection systems. *Annales des Telecommunications*, 55(7):361–378. ISSN 0003-4347.

Dempster A.P., Laird N., and Rubin D.B., 1977. Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Demšar Janez, 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30. ISSN 1532-4435.

Denning Dorothy E., May 1986. An Intrusion Detection Model. *Proceedings of the Seventh IEEE Symposium on Security and Privacy*, pages 119–131.

Devijver P., 1985. Baum's forward-backward algorithm revisited. *Pattern Recognition Letters*, 3:369–373.

Dietterich Thomas, 2000. Ensemble methods in machine learning. *Multiple Classifier Systems*, 1857:1–15.

Digalakis Vassilios V., 1999. Online adaptation of hidden Markov models using incremental estimation algorithms. *IEEE Transactions on Speech and Audio Processing*, 7 (3):253–261. ISSN 1063-6676.

Domingos Pedro, 2000. A unified bias-variance decomposition and its applications. *17th International Conference on Machine Learning*, pages 231–238. Morgan Kaufmann.

Dos Santos E. M, Sabourin R., and Maupin P., July 2008. Pareto analysis for the selection of classifier ensembles. *Genetic and Evolutionary Computation Conference (GECCO)*, pages 681–688, Atlanta, GA, USA, July 2008.

Du Ye, Wang Huiqiang, and Pang Yonggang, 2004. A hidden Markov models-based anomaly intrusion detection method. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, volume 5, pages 4348–4351, Hangzhou, China, 2004.

Eddy S. R., 1998. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763.

Elliott Robert J., 1994. Exact adaptive filters for Markov chains observed in gaussian noise. *Automatica*, 30(9):1399–1408. ISSN 0005-1098.

Elliott Robert J., Aggoun Lakhdar, and Moore John B., 1995. *Hidden Markov Models: Estimation and Control*. Springer-Verlag.

Ephraim Y. and Merhav N., 2002. Hidden Markov processes. *IEEE Transactions on Information Theory*, 48(6):1518–1569.

Ephraim Y. and Rabiner L.R., September 1990. On the relations between modeling approaches for information sources [speech recognition]. *IEEE Transactions on Information Theory*, 36(2):372–380.

Estevez-Tapiador Juan M., Garcia-Teodoro Pedro, and Diaz-Verdejo Jesus E., 2004. Anomaly detection methods in wired networks: A survey and taxonomy. *Computer Communications*, 27(16):1569–1584. ISSN 0140-3664.

Fan W., Miller M., Stolfo S., Lee W., and Chan P., 2004. Using artificial anomalies to detect unknown and known network intrusions. *Knowledge and Information Systems*, 6:507–527. ISSN 0219-1377.

Fan Wei, Chu Fang, Wang Haixun, and Yu Philip S., 2002. Pruning and dynamic scheduling of cost-sensitive ensembles. *Eighteenth national conference on Artificial intelligence*, pages 146–151, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

Fawcett Tom, 2004. ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Laboratories, Palo Alto, CA, USA.

Fawcett Tom, 2006. An introduction to ROC analysis. *Pattern Recognition Letter*, 27 (8):861–874. ISSN 0167-8655.

Flach Peter A. and Wu Shaomin, August 2005. Repairing concavities in ROC curves. *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 702–707. IJCAI.

Florez-Larrahondo German, Bridges Susan, and Hansen Eric A., 2005. Incremental estimation of discrete hidden Markov models based on a new backward procedure. *Proceedings of the National Conference on Artificial Intelligence*, 2:758–763.

Ford J.J. and Moore J.B., 1998a. Adaptive estimation of HMM transition probabilities. *IEEE Transactions on Signal Processing (USA)*, 46(5):1374–85. ISSN 1053-587X.

Ford J.J. and Moore J.B., 1998b. On adaptive HMM state estimation. *IEEE Transactions on Signal Processing (USA)*, 46(2):475–86. ISSN 1053-587X.

Forney G. David, March 2005. The viterbi algorithm: A personal history. *Viterbi Conference, University of Southern California, Los Angeles*.

Forrest S., Hofmeyr S., and Somayaji A., December 2008. The evolution of system-call monitoring. *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 418–430.

Forrest Stephanie, Hofmeyr Steven A., Somayaji Anil, and Longstaff Thomas A., 1996. A sense of self for Unix processes. *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128.

Foster James C., Osipov Vitaly, Bhalla Nish, and Heinen Niels, 2005. *Buffer Overflow Attacks: Detect, Exploit, Prevent*. Syngress. ISBN 1-932266-67-4.

Freund Yoav and Schapire Robert E., 1996. Experiments with a new boosting algorithm. *ICML 96*, pages 148–156.

Fürnkranz Johannes and Flach Peter, 2003. An analysis of rule evaluation metrics. *Proceedings of the 20th International Conference on Machine Learning*, volume 1, pages 202–209.

Gadelrab Mohammed El-Sayed, December 2008. *Evaluation of Intrusion Detection Systems*. PhD thesis, Université Toulouse III – Paul Sabatier.

Gael Jurgen Van, Saatci Yunus, Teh Yee Whye, and Ghahramani Zoubin, 2008. Beam sampling for the infinite Hidden Markov Model. *Proceedings of the 25th international conference on Machine learning*, pages 1088–1095, Helsinki, Finland, 2008. ACM.

Gao Bo, Ma Hui-Ye, and Yang Yu-Hang, 2002. HMMs (Hidden Markov Models) based on anomaly intrusion detection method. *Proceedings of 2002 International Conference on Machine Learning and Cybernetics*, 1:381–385.

Gao Fei, Sun Jizhou, and Wei Zunce, 2003. The prediction role of hidden Markov model in intrusion detection. *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 893–896, Montreal, Canada, 2003.

García Salvador and Herrera Francisco, December 2008. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

Garg Ashutosh and Warmuth Manfred K., September 2003. Inline updates for HMMs. *EUROSPEECH-2003*, pages 1005–1008.

Ghahramani Z., 2001. An introduction to hidden Markov models and Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42. ISSN 0218-0014.

Ghorbani Ali A., Lu Wei, Tavallaee Mahbod, Ghorbani Ali A., Lu Wei, and Tavallaee Mahbod, 2010. Intrusion response. Jajodia Sushil, editor, *Network Intrusion Detection and Prevention*, volume 47 of *Advances in Information Security*, pages 185–198. Springer US. ISBN 978-0-387-88771-5.

Ghosh Anup K., Schwartzbard Aaron, and Schatz Michael, 1999. Learning program behavior profiles for intrusion detection. *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, Berkeley, CA, USA, 1999. USENIX Association. ISBN 1-880446-37-5.

Gorodnichy Dmitry O., Dubrofsky Elan, Hoshino Richard, Khreich Wael, Granger Eric, and Sabourin Robert, April 2011. Exploring the upper bound performance limit of iris biometrics using score calibration and fusion. *(SSCI) CIBIM - 2011 IEEE Workshop on Computational Intelligence in Biometrics and Identity Management*, Paris, France, April 2011.

Gotoh Yoshihiko, Hochberg Michael M., and Silverman Harvey F., 1998. Efficient training algorithms for HMM's using incremental estimation. *IEEE Transactions on Speech and Audio Processing*, 6(6):539–548. ISSN 1063-6676.

Grice JA., Hughey R., and Speck D., 1997. Reduced space sequence alignment. *CABIOS*, 13(1):45–53.

Grossberg Stephen, 1988. Nonlinear neural networks: principles, mechanisms and architectures. *Neural Networks*, 1:17–61.

Gunawardana Asela and Byrne William, 2005. Convergence theorems for generalized alternating minimization procedures. *Journal of Machine Learning Research*, 6: 2049–2073. ISSN 1533-7928.

Haker Steven, Wells William M., Warfield Simon K., Talos Ion-Florin, Bhagwat Jui G., Goldberg-Zimring Daniel, Mian Asim, Ohno-Machado Lucila, and Zou Kelly H., 2005. Combining classifiers using their receiver operating characteristics and maximum likelihood estimation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 3749:506–514.

Hanley JA and McNeil BJ, 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36.

Hanley James A., 1988. The robustness of the "binormal" assumptions used in fitting ROC curves. *Medical Decision Making*, 8(3):197–203.

Hathaway R. J., 1986. Another interpretation of the EM algorithm for mixture distributions. *Statistics and Probability Letters*, 4:53–56.

Helali Rasha G. Mohammed, 2010. Data mining based network intrusion detection system: A survey. Sobh Tarek, Elleithy Khaled, and Mahmood Ausif, editors, *Novel*

Algorithms and Techniques in Telecommunications and Networking, pages 501–505. Springer Netherlands. ISBN 978-90-481-3662-9.

Hill J.M., Oxley M.E., and Bauer K.W., 2003. Receiver operating characteristic curves and fusion of multiple classifiers. *Proceedings of the 6th International Confenrence on Information Fusion*, volume 2, pages 815–822.

Ho Tin Kam, 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844. ISSN 0162-8828.

Ho Tin Kam, Hull J.J., and Srihari S.N., 1994. Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1): 66–75. ISSN 0162-8828.

Hoang X.D. and Hu J., 2004. An efficient Hidden Markov Model training scheme for anomaly intrusion detection of server applications based on system calls. *IEEE International Conference on Networks, ICON*, volume 2, pages 470–474, Singapore, 2004.

Hoang Xuan Dau, Hu Jiankun, and Bertok Peter, 2003. A multi-layer model for anomaly intrusion detection. *IEEE International Conference on Networks, ICON*, volume 1, pages 531–536.

Hofmeyr Steven A., Forrest Stephanie, and Somayaji Anil, 1998. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180.

Holst Ulla and Lindgren Georg, 1991. Recursive estimation in mixture models with Markov regime. *IEEE Transactions on Information Theory*, 37(6):1683–1690.

Hovland Geir E. and McCarragher Brenan J., 1998. Hidden Markov models as a process monitor in robotic assembly. *International Journal of Robotics Research*, 17(2): 153–168. ISSN 0278-3649.

Hu Jiankun, 2010. Host-based anomaly intrusion detection. Stavroulakis Peter and Stamp Mark, editors, *Handbook of Information and Communication Security*, pages 235–255. Springer Berlin Heidelberg. ISBN 978-3-642-04117-4.

Huang Jin and Ling C.X., 2005. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310. ISSN 1041-4347.

Huang Xuedong and Hon Hsiao-Wuen, 2001. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN 0130226165. Foreword By-Raj Reddy.

Ilgun Koral, Kemmerer R. A., and Porras Phillip A., March 1995. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21:181–199. ISSN 0098-5589.

Ingham Kenneth and Forrest Stephanie, 2005. Network firewalls. Vemuri V. Rao and Sreeharirao V., editors, *Enhancing Computer Security with Smart Technology*, pages 9–35. CRC Press.

Jha S., Tan K., and Maxion R.A., 2001. Markov chains, classifiers, and intrusion detection. *Proceedings of the Computer Security Foundations Workshop*, pages 206–219.

Juang Bing-Hwang, Levinson S., and Sondhi M., 1986. Maximum likelihood estimation for multivariate mixture observations of Markov chains (corresp.). *IEEE Transactions on Information Theory*, 32(2):307–309. ISSN 0018-9448.

Kershaw David, Gao Qigang, and Wang Hai, 2011. Anomaly-based network intrusion detection using outlier subspace analysis: A case study. Butz Cory and Lingras Pawan, editors, *Advances in Artificial Intelligence*, volume 6657 of *Lecture Notes in Computer Science*, pages 234–239. Springer Berlin / Heidelberg.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2009a. A comparison of techniques for on-line incremental learning of HMM parameters in anomaly detection. *Proceedings of the Second IEEE international conference on computational intelligence for security and defense applications (CISDA09)*, pages 1–8, Ottawa, Canada, 2009a. ISBN 978-1-4244-3763-4.

Khreich Wael, Granger Eric, Sabourin Robert, and Miri Ali, 2009b. Combining Hidden Markov Models for anomaly detection. *International Conference on Communications (ICC09)*, pages 1–6, Dresden, Germany, 2009b.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2010a. Boolean combination of classifiers in the ROC space. *20th International Conference on Pattern Recognition (ICPR10)*, pages 4299–4303, Istanbul, Turkey, 2010a.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, July 2010b. Iterative Boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs. *Pattern Recognition*, 43(8):2732–2752. ISSN 0031-3203.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2010c. On the memory complexity of the forward-backward algorithm. *Pattern Recognition Letters*, 31(2): 91–99. ISSN 0167-8655.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2011a. A survey of techniques for incremental learning of HMM parameters. *Information Sciences*. Accepted.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2011b. Incremental Boolean combination of classifiers. *10th International Workshop on Multiple Classifier Systems (MCS11)*, pages 340–349, Naples, Italy, 2011b.

Khreich Wael, Granger Eric, Miri Ali, and Sabourin Robert, 2011c. Adaptive ROC-based ensembles of HMMs applied to anomaly detection. *Pattern Recognition*. ISSN 0031-3203. Accepted.

Kittler J., March 1998. Combining classifiers: A theoretical framework. *Pattern Analysis & Applications*, 1(1):18–27.

Kivinen Jyrki and Warmuth Manfred K., 1997. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63. ISSN 0890-5401.

Koenig Sven and Simmons Reid G., 1996. Unsupervised learning of probabilistic models for robot navigation. *Proceedings - IEEE International Conference on Robotics and Automation*, 3:2301–2308. ISSN 1050-4729.

Konorski J., 2005. Solvability of a Markovian model of an IEEE802.11 LAN under a back-off attack. *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 491–8, Atlanta,GA,USA, 2005.

Kosoresow A.P. and Hofmeyer S.A., 1997. Intrusion detection via system call traces. *IEEE Software*, 14(5):35–42. ISSN 0740-7459.

Krishnamurthy V. and Moore J.B., 1993. On-line estimation of hidden Markov model parameters based on the Kullback-leibler information measure. *IEEE Transactions on Signal Processing, [see also IEEE Transactions on Acoustics, Speech, and Signal Processing]*, 41(8):2557–2573. ISSN 1053-587X.

Krishnamurthy V. and Yin George Gang, 2002. Recursive algorithms for estimation of hidden Markov models and autoregressive models with Markov regime. *IEEE Transactions on Information Theory*, 48(2):458–476. ISSN 0018-9448.

Krogh A., Brown M., Mian I. S., Sjolander K., and Haussler D., February 1994. Hidden Markov-models in computational biology applications to protein modeling. *JOURNAL OF MOLECULAR BIOLOGY*, 235(5):1501–1531.

Krogh A., Larsson B., von Heijne G., and Sonnhammer E. L. L., January 2001. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580.

Kruegel Christopher, Mutz Darren, Robertson William, and Valeur Fredrik, 2003. Bayesian event classification for intrusion detection. *Proceedings of the 19th Annual Computer Security Applications Conference*, ACSAC '03, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2041-3.

Krügel Christopher, Kirda E., Mutz D., Robertson W., and Vigna G., 2005. Automating mimicry attacks using static binary analysis. *Proceedings of the 14th USENIX Security Symposium*, pages 161–176, Baltimore, MD, USA, 2005.

Kumar Gulshan, Kumar Krishan, and Sachdeva Monika, 2010. The use of artificial intelligence based techniques for intrusion detection: a review. *Artificial Intelligence Review*, 34:369–387. ISSN 0269-2821.

Kuncheva Ludmila and Plumpton Catrin, 2008. Adaptive learning rate for online linear discriminant classifiers. *Joint IAPR International Workshops on Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition S+SSPR*, pages 510–519, Orlando, Florida, USA, 2008.

Kuncheva Ludmila I., 2002. A theoretical study on six classifier fusion strategies. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(2):281–286. ISSN 0162–8828.

Kuncheva Ludmila I., 2004a. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, Hoboken, NJ.

Kuncheva Ludmila I., 2004b. Classifier ensembles for changing environments. *Multiple Classifier Systems*, volume 3077, pages 1–15, Cagliari, Italy, 2004b. Springer-Verlag, LNCS.

Kuncheva Ludmila I. and Whitaker Christopher J., May 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207.

Kushner H.J. and Clark D.S., 1978. *Stochastic Approximation for Constrained and Unconstrained Systems*, volume 26. Springer-Verlag, New York.

Lam Tin and Meyer Irmtraud, 2010. Efficient algorithms for training the parameters of hidden markov models using stochastic expectation maximization (em) training and viterbi training. *Algorithms for Molecular Biology*, 5(1):38. ISSN 1748-7188.

Lane Terran and Brodley Carla E., 2003. An empirical study of two approaches to sequence learning for anomaly detection. *Machine Learning*, 51(1):73–107.

Lane Terran D., 2000. *Machine learning techniques for the computer security domain of anomaly detection*. PhD thesis, Purdue University, W. Lafayette, IN.

Langdon William B. and Buxton Bernard F., 2001. Evolving receiver operating characteristics for data fusion. *EuroGP '01: Proceedings of the 4th European Conference on Genetic Programming*, pages 87–96, London, UK, 2001. Springer-Verlag.

Lange Kenneth, 1995. A gradient algorithm locally equivalent to the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 57(2):425–437.

Lazarevic Aleksandar, Kumar Vipin, and Srivastava Jaideep, 2005. Intrusion detection: A survey. Kumar Vipin, Srivastava Jaideep, and Lazarevic Aleksandar, editors, *Managing Cyber Threats*, volume 5 of *Massive Computing*, pages 19–78. Springer US. ISBN 978-0-387-24230-9.

Lee Wenke and Xiang Dong, 2001. Information-theoretic measures for anomaly detection. *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143.

Lee Wenke, Stolfo Salvatore J., and Mok Kui W., 2000. Adaptive intrusion detection: A data mining approach. *Artificial Intelligence Review*, 14:533–567. ISSN 0269-2821.

LeGland F. and Mevel L., December 1995. Recursive identication of HMM's with observations in a finite set. *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 216–221, New Orleans, December 1995.

LeGland F. and Mevel L., December 1997. Recursive estimation in hidden Markov models. *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 4, pages 3468–3473, San Diego, CA, December 1997.

Leroux B.G., 1992. Maximum-likelihood estimation for hidden Markov models. *Stochastic Processes and its Applications*, 40:127–143.

Levinson S. E., Rabiner L. R., and Sondhi M. M., 1983. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell System Technical Journal*, 62:1035–1074.

Li Xiaolin, Parizeau M., and Plamondon R., 2000. Training hidden Markov models with multiple observations–A combinatorial method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):371–377. ISSN 0162-8828.

Li Z. and Evans R.J., 1992. Optimal filtering, prediction and smoothing of hidden Markov models. *Proceedings of the 31st IEEE Conference on Decision and Control*, volume 4, pages 3299–3304.

Liao Yihua and Vemuri V. Rao, 2002. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security*, 21(5):439–448. ISSN 0167-4048.

Lim S.Y. and Jones A., August 2008. Network anomaly detection system: The state of art of network behaviour analysis. *International Conference on Convergence and Hybrid Information Technology, ICHIT08*, pages 459–465.

Lindgren G., 1978. Markov regime models for mixed distributions and switching regressions. *Scandinavian Journal of Statistics*, 5:81–91.

Lindqvist Ulf and Porras Phillip A., 1999. Detecting computer and network misuse through the production-based expert system toolset (p-best). *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161.

Liporace L., 1982. Maximum likelihood estimation for multivariate observations of Markov sources. *IEEE Transactions on Information Theory*, 28(5):729–734.

Littlestone N. and Warmuth M. K., 1994. The weighted majority algorithm. *Information and Computation*, 108(2):212–261. ISSN 0890-5401.

Ljung L., 1977. Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, 22(4):551–575.

Ljung Lennart and Soderstrom Torsten, 1983. *Theory and Practice of Recursive Identification*. MIT Press, Boston.

MacKay D., 1997. Ensemble learning for hidden Markov models. Technical report, Cavendish Laboratory, Cambridge, UK.

Mann Tobias P., February 2006. Numerically stable hidden markov Model implementation. An HMM scaling tutorial.

Marceau Carla, 2000. Characterizing the behavior of a program using multiple-length n-grams. *Proceedings of the 2000 workshop on New security paradigms NSPW '00*, pages 101–110, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-260-3.

Margineantu Dragos D. and Dietterich Thomas G., 1997. Pruning adaptive boosting. *ICML*, pages 211–218.

Markou Markos and Singh Sameer, 2003a. Novelty detection: a review part 1: statistical approaches. *Signal Process.*, 83(12):2481–2497. ISSN 0165-1684.

Markou Markos and Singh Sameer, 2003b. Novelty detection: a review part 2: neural network based approaches. *Signal Process.*, 83(12):2499–2521. ISSN 0165-1684.

Martinez-Munoz Gonzalo, Hernandez-Lobato D., and Suraez Alberto, feb. 2009. An analysis of ensemble pruning techniques based on ordered aggregation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):245–259. ISSN 0162-8828.

Maxion R.A. and Tan K.M.C., 2000. Benchmarking anomaly-based detection systems. *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, pages 623–630.

McHugh J., Christie A., and Allen J., September/October 2000. Defending yourself: the role of intrusion detection systems. *IEEE Software*, 17(5):42–51. ISSN 0740-7459.

Metz C.E., 1978. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8: 283–298.

Meyer Irmtraud M. and Durbin Richard, 2004. Gene structure conservation aids similarity based gene prediction. *Nucleic Acids Research*, 32(2):776–783.

Michael C. C. and Ghosh Anup, August 2002. Simple, state-based approaches to program-based anomaly detection. *ACM Trans. Information System Security*, 5:203–237. ISSN 1094-9224.

Miklos Istvan and Meyer Irmtraud M., 2005. A linear memory algorithm for baum-welch training. *BMC Bioinformatics*, 6:231.

Mitnick Kevin D. and Simon William L., 2005. *The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders and Deceivers.* Wiley.

Mizuno J., Watanabe T., Ueki K., Amano K., Takimoto E., and Maruoka A., 2000. On-line estimation of hidden Markov model parameters. *Proceedings of Third International Conference on Discovery Science, DS 2000*, 1967:155–169.

Mongillo Gianluigi and Deneve Sophie, 2008. Online learning with hidden Markov models. *Neural Computation*, 20(7):1706–1716.

Muhlbaier M., Topalis A., and Polikar R., July 2004. Incremental learning from unbalanced data. *Proc. of the Internaltional Joint Conference on Neural Networks (IJCNN)*, pages 1057–1062, Budapest, Hungary, July 2004.

Narciso Tan Lau, 1993. *Adaptive channel/code matching.* PhD thesis, University of Southern California, California, United States.

Neal R. M. and Hinton G. E., 1998. A new view of the EM algorithm that justifies incremental, sparse and other variants. Jordan M. I., editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers.

Neyman J. and Pearson E. S., 1933. On the problem of the most efficient tests of statistical hypotheses. *Royal Society of London Philosophical Transactions Series A*, 231:289–337.

Ng Brenda, October 2006. Survey of anomaly detection methods. Technical Report UCRL-TR-225264, Lawrence Livermore National Laboratory, Livermore, CA.

Nocedal Jorge and Wright Stephen J., 2006. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition.

Northcutt Stephen and Novak Judy, 2002. *Network Intrusion Detection: An Analyst's Handbook.* New Riders Publishing, Thousand Oaks, CA, USA, 3rd edition. ISBN 0735712654.

Ott G., 1967. Compact encoding of stationary markov sources. *Information Theory, IEEE Transactions on*, 13(1):82–86. ISSN 0018-9448.

Ourston D., Matzner S., Stump W., and Hopkins B., January 2003. Applications of hidden Markov models to detecting multi-stage network attacks. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences.*

Oxley M.E., Thorsen S.N., and Schubert C.M., 2007. A Boolean Algebra of receiver operating characteristic curves. *10th International Conference on Information Fusion*, pages 1–8.

Oza Nikunj C. and Russell Stuart, January 2001. Online bagging and boosting. Jaakkola Tommi and Richardson Thomas, editors, *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 105–112, Key West, Florida. USA, January 2001. Morgan Kaufmann.

Parampalli Chetan, Sekar R., and Johnson Rob, 2008. A practical mimicry attack against powerful system-call monitors. *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 156–167, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-979-1.

Partalas Ioannis, Tsoumakas Grigorios, and Vlahavas Ioannis, 2008. Focused ensemble selection: A diversity-based method for greedy ensemble selection. *Proceeding of the 2008 conference on ECAI 2008*, pages 117–121, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. ISBN 978-1-58603-891-5.

Paxson Vern, December 1999. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463. ISSN 1389-1286.

Peng Tao, Leckie Christopher, and Ramamohanarao Kotagiri, April 2007. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)*, 39(1):1–42. ISSN 0360-0300.

Pepe Margaret Sullivan and Thompson Mary Lou, 2000. Combining diagnostic test results to increase accuracy. *Biostat*, 1(2):123–140.

Polikar R., Upda L., Upda S.S., and Honavar V., 2001. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 31(4):497–508.

Polikar Robi, 2006. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45.

Polyak B. T., 1991. New method of stochastic approximation type. *Automation Remote Contr.*, 7:937–946.

Poritz A.B., 1988. Hidden Markov models: a guided tour. *International Conference on Acoustics, Speech, and Signal Processing ICASSP-88.*, volume 1, pages 7–13.

Porras Phillip A. and Neumann Peter G., 1997. EMERALD: Event monitoring enabling responses to anomalous live disturbances. *In Proceedings of the 20th National Information Systems Security Conference*, pages 353–365.

Proctor Paul E., 2000. *Practical Intrusion Detection Handbook*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition. ISBN 0130259608.

Provost Foster and Fawcett Tom, 1997. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48, Menlo Park, CA, 1997. AAAI Press.

Provost Foster J. and Fawcett Tom, 2001. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231.

Provost Foster J., Fawcett Tom, and Kohavi Ron, 1998. The case against accuracy estimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann Publishers Inc.

Ptacek Thomas H. and Newsham Timothy N., 1998. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical Report T2R-0Y6, Secure Networks, Calgary, Canada.

Qiao Y., Xin X.W., Bin Y., and Ge S., 2002. Anomaly intrusion detection method based on HMM. *Electronics Letters*, 38(13):663–664.

Qin Feng, Auerbach Anthony, and Sachs Frederick, 2000. A direct optimization approach to hidden markov modeling for single channel kinetics. *Biophysical Journal*, 79(4): 1915–1927.

Rabiner L., 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rash Michael, Orebaugh Angela D., Clark Graham, Pinkard Becky, and Babbin Jake, 2005. *Intrusion Prevention and Active Response: Deployment Network and Host IPS*. Syngress.

Raudys Å arunas and Roli Fabio, 2003. The behavior knowledge space fusion method: Analysis of generalization error and strategies for performance improvement. *Multiple Classifier Systems*, 2709:55–64.

Raviv J., 1967. Decision making in markov chains applied to the problem of pattern recognition. *Information Theory, IEEE Transactions on*, 13(4):536–551. ISSN 0018-9448.

Rittscher Jens, Kato Jien, Joga Sébastien, and Blake Andrew, 2000. A probabilistic background model for tracking. *Proceedings of the 6th European Conference on Computer Vision-Part II, ECCV00*, pages 336–350, London, UK, 2000. Springer-Verlag. ISBN 3-540-67686-4.

Roesch Martin, 1999. Snort–lightweight intrusion detection for networks. *13th Systems Administration Conference (LISA)*, pages 229–238.

Rokach L., Maimon O., and Arbel R., 2006. Selective voting–getting more for less in sensor fusion. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(3):329–350.

Rokach Lior, February 2010. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39.

Roli F., Fumera G., and Kittler J., 2002. Fixed and trained combiners for fusion of imbalanced pattern classifiers. *Information Fusion, 2002. Proceedings of the Fifth International Conference on*, volume 1, pages 278–284 vol.1.

Ruta Dymitr and Gabrys Bogdan, October 2002. A theoretical analysis of the limits of majority voting errors for multiple classifier systems. *Pattern Analysis & Applications*, 5(4):333–350.

Ruta Dymitr and Gabrys Bogdan, March 2005. Classifier selection for majority voting. *Information Fusion*, 6(1):63–81. ISSN 1566-2535.

Ryden T., 1998. Asymptotic efficient recursive estimation for incomplete data models using the observed information. *Metrika*, 44:119–145.

Ryden Tobias, 1994. Consistent and asymptotically normal parameter estimates for hidden Markov models. *Annals of Statistics*, 22(4):1884–1895.

Ryden Tobias, February 1997. On recursive estimation for hidden Markov models. *Stochastic Processes and their Applications*, 66(1):79–96.

Salem Malek Ben, Hershkop Shlomo, and Stolfo Salvatore J., 2008. A survey of insider attack detection research. Jajodia Sushil, Stolfo Salvatore J., Bellovin Steven M., Keromytis Angelos D., Hershkop Shlomo, Smith Sean W., and Sinclair Sara, editors, *Insider Attack and Cyber Security*, volume 39 of *Advances in Information Security*, pages 69–90. Springer US. ISBN 978-0-387-77322-3.

Saltzer J.H. and Schroeder M.D., September 1975. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278 –1308. ISSN 0018-9219.

Scarfone Karen and Mell Peter, February 2007. Guide to intrusion detection and prevention systems (IDPS). Recommendations of the National Institute of Standards and Technology sp800-94, NIST, Technology Administration, Department of Commerce, USA, 2007.

Schraudolph N.N., 1999. Local gain adaptation in stochastic gradient descent. *Ninth International Conference on Artificial Neural Networks, ICANN99*, volume 2, pages 569–574.

Scott M. J. J., Niranjan M., and Prager R. W, September 1998. Realisable classifiers: Improving operating performance on variable cost problems. Lewis Paul H. and

Nixon Mark S., editors, *Proceedings of the Ninth British Machine Vision Conference*, volume 1, pages 304–315, University of Southampton, UK, September 1998.

Sekar R., Bendre M., Dhurjati D., and Bollineni P., 2001. A fast automaton-based method for detecting anomalous program behaviors. *IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

Shen Changyu, 2008. On the principles of believe the positive and believe the negative for diagnosis using two continuous tests. *Journal of Data Science*, 6:189–205.

Shue L., Anderson D.O., and Dey S., 1998. Exponential stability of filters and smoothers for hidden markov models. *Signal Processing, IEEE Transactions on*, 46(8):2180–2194. ISSN 1053-587X.

Silberschatz Avi, Galvin Peter Baer, and Gagne Greg, July 2008. *Operating System Concepts*. John Wiley & Sons.

Singer Yoram and Warmuth Manfred K., 1996. Training algorithms for hidden Markov models using entropy based distance functions. *Neural Information Processing Systems*, pages 641–647.

Sivaprakasam S. and Shanmugan Sam K., 1995. A forward-only recursion based HMM for modeling burst errors in digital channels. *Global Telecommunications Conference, 1995. GLOBECOM '95., IEEE*, volume 2, pages 1054–1058.

Slingsby P.L., 1992. Recursive parameter estimation for arbitrary hidden Markov models. *IFAC Adavptive Systems in Control and Signal Processing*, Grenoble, France, 1992.

Smyth Padhraic, Heckerman David, and Jordan Michael, 1997. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9: 227–269.

Somayaji Anil B., July 2002. *Operating System Stability and Security through Process Homeostasis*. PhD thesis, University of New Mexico.

Stakhanova Natalia, Basu Samik, and Wong Johnny, 2007. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1 (1/2):169–184.

Stenger B., Ramesh V., Paragios N., Coetzee F., and Buhmann J.M., 2001. Topology free hidden Markov models: Application to background modeling. *Proceedings of the IEEE International Conference on Computer Vision*, 1:294–301.

Stiller J.C., 2003. Adaptive online learning of generative stochastic models. *Complexity (USA)*, 8(4):95–101. ISSN 1076-2787.

Stiller J.C. and Radons G., 1999. Online estimation of hidden Markov models. *IEEE Signal Processing Letters (USA)*, 6(8):213–15. ISSN 1070-9908.

Tan K.M.C. and Maxion R.A., 2002. "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector. *IEEE Symposium on Security and Privacy*, pages 188–201.

Tan K.M.C. and Maxion R.A., 2003. Determining the operational limits of an anomaly-based intrusion detector. *IEEE Journal on Selected Areas in Communications*, 21 (1):96–110.

Tan K.M.C. and Maxion R.A., 2005. The effects of algorithmic diversity on anomaly detector performance. *International Conference on Dependable Systems and Networks (DSN)*, pages 216–225.

Tan Kymie M. C., Killourhy Kevin S., and Maxion Roy A., October 2002. Undermining an anomaly-based intrusion detection system using common exploits. *Proceedings of the 5th international conference on Recent advances in intrusion detection*, volume 2516 of *RAID'02*, pages 54–73, Berlin, Heidelberg, October 2002. Springer-Verlag.

Tandon G. and Chan P., 2006. On the learning of useful system call attributes for host-based anomaly detection. *International Journal on Artificial Intelligence Tools*, 15 (6):875–892.

Tao Qian and Veldhuis Raymond, 2008. Threshold-optimized decision-level fusion and its application to biometrics. *Pattern Recognition*, 41(5):852–867. ISSN 0031-3203.

Tarnas Christopher and Hughey Richard, 1998. Reduced space hidden markov model training. *Bioinformatics*, 14:401–406.

Thomopoulos S.C.A., Viswanathan R., and Bougoulias D.K., 1989. Optimal distributed decision fusion. *IEEE Transactions on Aerospace and Electronic Systems*, 25(5): 761–765.

Titterington D. M., 1984. Recursive parameter estimation using incomplete data. *Journal of the Royal Statistical Society, Series B (Methodological)*, 46(2):257–267.

Tosun Umut, 2005. Hidden Markov models to analyze user behaviour in network traffic. Technical report, Bilkent University 06800 Bilkent, Ankara, Turkey.

Tsai Chih-Fong, Hsu Yu-Feng, Lin Chia-Ying, and Lin Wei-Yang, December 2009. Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36(10):11994–12000. ISSN 0957-4174.

Tsoumakas Grigorios, Partalas Ioannis, and Vlahavas Ioannis, 2009. An ensemble pruning primer. *Applications of Supervised and Unsupervised Ensemble Methods*, 245:1–13.

Tucker C.J., Furnell S.M., Ghita B.V., and Brooke P.J., 2007. A new taxonomy for comparing intrusion detection systems. *Internet Research*, 17:88–98.

Tulyakov Sergey, Jaeger Stefan, Govindaraju Venu, and Doermann David, 2008. Review of classifier combination methods. Simone Marinai Hiromichi Fujisawa, editor, *Studies in Computational Intelligence: Machine Learning in Document Analysis and Recognition*, pages 361–386. Springer.

Turner Rolf, 2008. Direct maximization of the likelihood of a hidden Markov model. *Computational Statistics and Data Analysis*, 52(9):4147–4160. ISSN 0167-9473.

Ulaş Aydin, Semerci Murat, Yildiz Olcay Taner, and Alpaydin Ethem, 2009. Incremental construction of classifier and discriminant ensembles. *Information Sciences*, 179 (9):1298–1318.

Vaccaro H.S. and Liepins G.E., May 1989. Detection of anomalous computer session activity. *IEEE Symposium on Security and Privacy*, pages 280–289.

Van Erp Merijn and Schomaker Lambert, 2000. Variants of the borda count method for combining ranked classifier hypotheses. *Seventh International Workshop on Frontiers in Handwriting Recognition*, Amsterdam, 2000.

Varshney P. K., 1997. *Distributed detection and data fusion*. Springer-Verlag, New York.

Venkataramani Krithika and Kumar B., 2006. Role of statistical dependence between classifier scores in determining the best decision fusion rule for improved biometric verification. *Multimedia Content Representation, Classification and Security*, 4105: 489–496.

Vigna G. and Kruegel C., December 2005. Host-based intrusion detection systems. Bigdoli H., editor, *Handbook of Information Security*, volume III. Wiley.

Viterbi A. J., April 1967. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Infromation Theory*, IT-13: 260–269.

Wagner David and Dean Drew, 2001. Intrusion detection via static analysis. *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2001. IEEE Computer Society.

Wagner David and Soto Paolo, 2002. Mimicry attacks on host-based intrusion detection systems. *Proceedings of the 9th ACM conference on Computer and communications security*, pages 255–264, Washington, DC, United States, 2002.

Walter S. D., 2005. The partial area under the summary ROC curve. *Statistics in Medicine*, 24(13):2025–2040.

Wang Panhong, Shi Liang, Wang Beizhan, Wu Yuanqin, and Liu Yangbin, August 2010. Survey on HMM based anomaly intrusion detection using system calls. *Computer Science and Education (ICCSE), 2010 5th International Conference on*, pages 102–105.

Wang Shaojun and Zhao Yunxin, December 2006. Almost sure convergence of Titterington's recursive estimator for mixture models. *Statistics and Probability Letters*, 76 (18):2001–2006.

Wang Wei, Guan Xiao-Hong, and Zhang Xiang-Liang, 2004. Modeling program behaviors by Hidden Markov Models for intrusion detection. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 5:2830–2835.

Warrender Christina, Forrest Stephanie, and Pearlmutter Barak, 1999. Detecting intrusions using system calls: Alternative data models. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 133–45, Oakland, CA, USA, 1999.

Weinstein E. M. Feder and Oppenheim A. V., 1990. Sequential algorithms for parameter estimation based on the Kullback-Leibler information measure. *IEEE Transactions on Acoustics Speech and Signal Processing*, 38(9):1652–1654.

Wolpert David H., 1992. Stacked generalization. *Neural Networks*, 5:241–259.

Wolpert D.H. and Macready W.G., apr 1997. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82. ISSN 1089-778X.

Wu C. F. Jeff, March 1983. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11(1):95–103.

Yeung Dit-Yan and Ding Yuxin, 2003. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36(1):229–243.

Zhang Dong D., Zhou Xia-Hua, Freeman Jr. Daniel H., and Freeman Jean L., 2002. A non-parametric method for the comparison of partial areas under ROC curves and its application to large health care data sets. *Statistics in Medicine*, 21(5):701–715.

Zhang Xiaoqiang, Fan Pingzhi, and Zhu Zhongliang, 2003. A new anomaly detection method based on hierarchical HMM. *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, pages 249–252.

Zhang Zhihao and Zhou Jie, September 2010. Transfer estimation of evolving class priors in data stream classification. *Pattern Recognition*, 43(9):3151–3161. ISSN 0031-3203.

Zucchini Walter and MacDonald Iain, 2009. *Hidden Markov and Other Models for Discrete-valued Time Series: An Introduction Using R.* Monographs on statistics and applied probability, 110. Chapman & Hall.